Differentiable Physically Based Rendering: Algorithms, Systems and Applications

by Merlin Nimier-David

École Polytechnique Fédérale de Lausanne

Switzerland

October 2022

Abstract

Physically based rendering methods can create photorealistic images by simulating the propagation and interaction of light in a virtual scene. Given a scene description including the shape of objects, participating media, material properties, etc., the simulation computes an image representing the radiance reaching the sensor.

This thesis, however, pursues the corresponding inverse problem: given observations such as pictures of a scene, we want to recover a plausible description of its components. Unfortunately, the rendering process is typically far too complex to invert analytically. We therefore turn to iterative gradient-based approximations of the inverse, that require efficiently estimating gradients of an objective function with respect to the scene parameters of interest.

The first part of this thesis is dedicated to algorithms for gradient estimation. We consider the usage of automatic differentiation and examine the associated tradeoffs. Next, we introduce *radiative backpropagation*, an adjoint method that casts the gradient estimation problem into a modified light transport problem, unlocking vastly more efficient implementations. For the case of participating media, we propose *differential ratio tracking*, a sampling technique that addresses the bias and high variance found in existing gradient estimators.

In the second part, we focus on the design of systems to support effective differentiable rendering research, including the efficient implementation of the algorithms above. We describe the architecture and features of Mitsuba 2, an open-source retargetable physically based renderer. Mitsuba 2 supports different representations of colors (RGB, spectral, polarized), computing platforms (scalar, CPU vectorized, GPU), and numerical precision, within a single codebase. Importantly, automatic differentiation can be applied throughout the system. Then, we extend this design by applying an automatic conversion from wavefront-style rendering to a megakernel-based approach, leveraging just-in-time compilation. We obtain a fast, flexible and memory-efficient framework for primal and differentiable physically based rendering.

Finally, the third part showcases three applications of differentiable physically based rendering: caustic design, inverse volume rendering, and material & lighting estimation in real indoor scenes. In all cases, special care is taken to avoid sub-optimal local minima due to the ambiguous and non-convex nature of the reconstruction problems.

Keywords: physically based rendering, differentiable rendering, inverse rendering, gradient descent, adjoint method, automatic differentiation, participating media, caustics.

Acknowledgements

First and foremost, I am immensely grateful to my advisor, Prof. Wenzel Jakob. His constant availability, guidance, patience, generosity and deep knowledge made this PhD an unparalleled learning and growth experience. I am thankful that he trusted me and gave me this unique opportunity.

I am grateful to my thesis committee members Prof. Martin Jaggi, Prof. Tzu-Mao Li, Prof. Mark Pauly, and Prof. Shuang Zhao for their commitment and insightful comments. Having world-class domain experts review this work was a great privilege.

I would also like to thank my co-authors, who significantly contributed to the research presented in this thesis: Zhao Dong, Anton Kaplanyan, Alexander Keller, Thomas Müller, Benoît Ruiz, Sébastien Speierer, Delio Vicini, and Tizian Zeltner.

Since the start of my studies, I had the privilege to meet mentors who were instrumental in shaping my trajectory. I am grateful for the invaluable advice offered by Matthieu Bacconnier, Prof. Florent de Dinechin, Alexandru Ichim, Prof. Mark Pauly, Peyman Milanfar, Anton Kaplanyan, and Alexander Keller.

I was lucky to be surrounded by the talented (past and present) members of the Realistic Graphics Lab at EPFL: Miguel Crespo, Guillaume Loubet, Baptiste Nicolet, Pauline Raffestin, Nicolas Roussel, Sébastien Speierer, Delio Vicini, Mengqi Xia, Tizian Zeltner, and Ziyi Zhang. They were always there for technical discussions, working through difficult concepts together, and providing mutual support and memes. Going through the paper deadlines and the PhD's ups and downs together made it so much easier.

More broadly, EPFL and my fellow PhD students in the IC department created an amazing environment in which to learn and grow.

When away on internships, I had the privilege to be hosted by teams full of smart and kind individuals who made me feel welcome and taught me a lot: the Graphics & Vision Group at Cornell, the Computational Imaging team at Google Research, the Graphics & Surreal teams at Facebook Reality Labs, and the Graphics, Communications & Machine Learning team at NVIDIA.

Over the course of my PhD studies, I had the opportunity to supervise projects driven by many talented EPFL students. I thank them for their hard work and dedication, in particular Thomas Ibanez and Damien Martin. It was a pleasure to co-supervise projects with Bogdan Kulynych.

The research in this thesis was supported by funding from the Swiss National Science Foundation (SNSF) and a Facebook Graduate Fellowship. It was also made possible by numerous free and open-source tools and libraries, including Blender [1], Clang [2], Homebrew, imageio, Jupyter, LaTeX, Markdown, Matplotlib [3], Mosh, NumPy [4], OpenEXR, pandas [5, 6], Publish or Perish [7], pybind11 [8], pytest [9], Python, SciPy [10], Seaborn [11], and tqdm. Tizian Zeltner provided the template layout used in this thesis. I also owe a lot to the many artists who generously make their work available under permissive licenses.

I feel so lucky to have met Salma, who accompanied me on these (and future) journeys. She helped me keep my sanity, balance and motivation even at times when nothing was working in my research.

Finally, my parents have given me invaluable support throughout my studies, for which I am deeply thankful. They taught me the value of studying for myself as opposed to working to please someone else. They also enabled me to pursue my passion for computers, even when that meant playing around with their main working tool or spending countless hours at my desk.

Figure 1.2 uses the “Bed Classic” scene by jiraniano on BlendSwap.
Figure 2.7 uses the “Bed” model by sadaj72 on BlendSwap.
Tables 3.1, 4.3 and 7.2 use the “The Wooden Staircase” scene by Wig42 on BlendSwap.
Figure 4.1 uses a scene modeled by Olesya Jakob.
Figures 4.1 and 4.5 use a Mars texture made available by Solar System Scope.
Figures 4.7, 7.1 and 10.3 use the “smoke2” model from OpenVDB’s sample models.
Figures 5.1, 5.10 and 10.6 use the “High-Res Smoke Plume” model from JangaFX.
Figures 5.3, 5.8, 10.2 and 10.8 use the “Astronaut - EMU suit” model from jgilhutton on Blendswap.
Figures 5.5 and 10.1 use the Cloud Data Set from Walt Disney Animation Studios and an ocean environment map by Antoan Shiyachki.
Figure 7.1 uses scenes (a) and (c) modeled by Tizian Zeltner, scene (b) is “The Grey &

White Room" by Wig42 on BlendSwap, and scene (d) was modeled by Delio Vicini.
Figure 10.4 uses the "Manned sci fi Rover" model by vajrablue on BlendSwap and the "Trees" model by Zuendholz on BlendSwap.
Figures 10.6 and 10.8 uses the "Dust Devil (Tornado)" model from JangaFX.
Figure 11.2 uses the "Flashlight (batteries)" model by richanatario on BlendSwap.
Figure 11.4 uses the "Apples for Cycles" model by Alex Telford on BlendSwap.
Figure 11.14 uses scene edits by Matthew Chapman.
Many of the scenes additionally use environment maps from PolyHaven.

À Franck, Marie et Olivier.

Table of Contents

1 Introduction **1**

1.1 Overview . . . 3

1.2 List of publications . . . 5

2 Background **6**

2.1 Monte Carlo integration . . . 6

2.1.1 Example problem . . . 6

2.1.2 Estimator . . . 7

2.1.3 Importance sampling . . . 7

2.1.4 Bias and variance . . . 8

2.1.5 Russian roulette . . . 10

2.1.6 Multiple importance sampling . . . 10

2.1.7 Pseudo- and quasi-random numbers . . . 11

2.2 Physically based rendering . . . 12

2.2.1 Setting and scope . . . 13

2.2.2 Rendering equations for the surface case . . . 15

2.2.3 Participating media . . . 16

2.2.4 Rendering algorithms . . . 17

2.3 Automatic differentiation . . . 21

2.4 Adjoint methods . . . 23

2.5 Compilers and domain-specific languages . . . 24

2.6 Differentiable rendering . . . 26

2.6.1 Problem statement . . . 26

2.6.2 Neutron transport . . . 26

2.6.3 Differentiable rasterization . . . 27

2.6.4 Physically based differentiable rendering . . . 28

2.6.5 Differentiable rendering and machine learning . . . 31

2.6.6 Applications of differentiable rendering . . . 31

I Algorithms **33**

3 Differentiable rendering with automatic differentiation **37**

3.1 Algorithm . . . 37

3.2 Correctness 38
3.3 Memory usage 40
3.4 Selective usage of AD 41

4 Radiative Backpropagation 44
4.1 Introduction 44
4.2 Radiative transfer 44
4.3 Method 46
4.3.1 Differential radiative transfer 46
4.3.2 Optimization using differential transport 47
4.3.3 Adjoint radiance 49
4.3.4 Operator formulation 50
4.3.5 Volumetric transport 51
4.3.6 Sampling strategies for differential rendering 53
4.3.7 Radiative backpropagation path tracing 54
4.3.8 Worse is better? Acceleration using biased gradients. 56
4.4 Evaluation 58
4.4.1 Validation 58
4.4.2 Performance 58
4.4.3 Texture optimization 60
4.4.4 Volume optimization 61
4.5 Conclusion 65

5 Unbiased Inverse Volume Rendering with Differential Trackers 67
5.1 Introduction 67
5.2 Background 70
5.2.1 Volumetric path tracing 70
5.2.2 Path replay backpropagation 71
5.3 Issues with free-flight based gradient estimation 72
5.3.1 Differentiating the radiative transfer equation 73
5.3.2 Free-flight based gradient estimators 74
5.4 Differential ratio tracking 77
5.4.1 Unbiased estimators 77
5.4.2 Sampling proportionally to transmittance 78
5.4.3 Differential ratio tracking 83
5.4.4 Multiple importance sampling 87

5.4.5 Preserving linear time complexity . . . 87
5.5 Evaluation . . . 88
5.5.1 Correctness and variance . . . 88
5.5.2 Role of the optimizer . . . 92
5.5.3 Implementation and performance . . . 92
5.6 Conclusion . . . 95

II Systems **96**

6 Systems for physically based differentiable rendering **97**
6.1 Scale of rendering systems . . . 97
6.2 Differentiating through renderers . . . 98

7 Mitsuba 2 **100**
7.1 Introduction . . . 100
7.2 Background . . . 101
7.2.1 Template metaprogramming . . . 101
7.2.2 Expression templates . . . 103
7.2.3 The Enoki library . . . 103
7.3 System design . . . 110
7.3.1 Architecture . . . 110
7.3.2 Language bindings . . . 114
7.3.3 Feature set . . . 115
7.3.4 Challenges . . . 116
7.4 Evaluation . . . 117
7.5 Conclusion . . . 117

8 From wavefront to megakernel **121**
8.1 Wavefront rendering in Mitsuba 2 . . . 122
8.2 Megakernel translation . . . 123
8.2.1 Wavefronts and megakernels . . . 123
8.2.2 Transitioning to a megakernel . . . 124
8.3 Performance evaluation . . . 126
8.4 Conclusion . . . 129

III Applications 131

9 Caustic optimization 133

9.1 Surface displacements 133

9.2 Gradient-index optics 137

9.3 Conclusion 139

10 Inverse volume rendering 140

10.1 Combating local minima 140

10.1.1 Source of local minima 141

10.1.2 Emissive volume initialization 142

10.1.3 Inverse volume rendering 143

10.2 Conclusion 151

11 Inverse Rendering of Real Rooms 152

11.1 Introduction 152

11.2 Related work 154

11.3 Method 156

11.3.1 Input data 156

11.3.2 Inductive bias & modeling assumptions 159

11.3.3 Texture-space sampling for variance reduction 162

11.3.4 Optimization details 163

11.4 Results 166

11.4.1 Reconstruction of real captured scenes 166

11.4.2 Implementation 166

11.4.3 Applications 168

11.5 Conclusion 169

12 Conclusion 174

A Inverse Volume Rendering with the Null-Scattering Integral Formulation 178

B Inverse Rendering of Real Rooms — additional results 181

B.1 Comparison to prior work 181

B.2 Ablation study 181

References 185

List of Figures

1.1 Characterization of our work 2
1.2 Inverse rendering diagram 3
1.3 Topics overview 4

2.1 Example of consistent and unbiased estimators 9
2.2 Monte Carlo integration example 12
2.3 Examples of real-world light transport effects 13
2.4 Typical scene components 18
2.5 Handling discontinuities in rasterization 28
2.6 Discontinuities in path tracing 29
2.7 Inverse rendering setup overview 35

3.1 Effect of correlation on gradient correctness 39
3.2 AD misses gradients at discontinuities 40
3.3 Correctness of Russian roulette gradients (detached pdf) 41
3.4 Example AD graph built by Mitsuba 2 43

4.1 Texture optimization, Globe scene 45
4.2 Radiative backpropagation pipeline diagram 48
4.3 Validation of radiative backpropagation gradients 59
4.4 Radiative backpropagation runtime breakdown 60
4.5 Radiative backpropagation application: texture optimization 62
4.6 Radiative backpropagation application: 3D printing 63
4.7 Radiative backpropagation application: volume optimization 64

5.1 Unbiased inverse volume rendering with differential trackers 68
5.2 Unbiased inverse volume rendering with differential trackers 69
5.3 Differential ratio tracking results preview 72
5.4 Delta tracking for the adjoint problem (diagram) 73
5.5 Bias of free-flight based sampling methods 75
5.6 Transmittance estimators and the corresponding differential trackers . . 79
5.7 Differential delta and ratio tracking (diagram) 81
5.8 Differential ratio tracking gradients validation 89
5.9 Validation of differential ratio tracking against analytic gradients 91

5.10 Differential ratio tracking ablation study 93

7.1 Example applications of Mitsuba 2 100
7.2 Volume gradient image computed with Mitsuba 2 107
7.3 Mitsuba 2 architecture diagram 111
7.4 Mitsuba 2 data structure instantiation 113
7.5 Code generation example: importance sampling of GGX 114

8.1 Automatic conversion of wavefront-style programs to a megakernel . . . 121
8.2 Recording Mitsuba 2 plugins for megakernel conversion 125
8.3 Relative runtime of megakernel-style radiative backpropagation compared to Mitsuba 2 128

9.1 Experimental setup for caustic design experiments (diagram) 133
9.2 Caustic optimization results 136
9.3 Example gradient-index lenses 137

10.1 Inverse volume rendering of clouds 140
10.2 Volume reconstruction with our novel initialization scheme 141
10.3 Presence of local minima in volume optimization 142
10.4 Differential ratio tracking optimization results (solid objects) 146
10.5 Differential ratio tracking optimization re-rendering loss (solid objects) . 147
10.6 Differential ratio tracking optimization results (volumes) 148
10.7 Differential ratio tracking optimization re-rendering loss (volumes) . . . 149
10.8 Differential ratio tracking relighting results 150

11.1 Inverse rendering of real rooms 152
11.2 Limitations of simplified solutions for scene reconstruction 153
11.3 Scene reconstruction pipeline diagram 157
11.4 Importance of accounting for global illumination in inverse renderings . 158
11.5 Example frames from the Replica dataset captures 159
11.6 Local minima in unconstrained inverse rendering (emission) 160
11.7 Local minima in unconstrained inverse rendering (roughness) 161
11.8 Uneven observation frequency in handheld video dataset 163
11.9 Texture-space sampling scheme 164
11.10 Re-rendering results (side-by-side) 167
11.11 Texture-space visualization of the recovered albedo textures 168
11.12 Rendering from novel viewpoints 170

11.13 Rendering from novel viewpoints (continued) 171
11.14 Relighting and scene editing results 172
11.15 Relighting a reconstructed indoor scene 172
11.16 Limitations of scene reconstruction 173

B.1 Reconstruction results, comparison to previous method 182
B.2 Inverse rendering ablation study (quantitative) 183
B.3 Inverse rendering ablation study (qualitative) 183

List of Listings

1 Path tracing pseudocode 19
2 Radiative backpropagation pseudocode (loop over samples) 55
3 Radiative backpropagation pseudocode (per sample handling) 55
4 Weighted reservoir sampling pseudocode 84
5 Differential ratio tracking pseudocode 85
6 Differential residual ratio tracking pseudocode 86
7 Automatic differentiation with Enoki 107
8 Mitsuba 2's `SurfaceInteraction` data structure 112
9 Example usage of Mitsuba 2's features through the Python bindings . . . 115
10 Excerpt from the implementation of Mitsuba 2's diffuse BSDF plugin . . 118
11 Automatic kernel extraction pseudocode 126
12 Kernel extraction with automatic differentiation 127
13 Enoki-based Eikonal equation solver with a "leapfrog" discretization . . . 138

1 | Introduction

Two main types of *rendering algorithms*, that generate images from a user-provided description of the scene to be shown, are in widespread use.

Rasterization focuses on realtime performance and has long benefited from hardware support in GPUs. Its approach of rendering an image into individual pixel contributions makes it difficult to achieve physical realism, and can involve taking many shortcuts. Rasterization is used in video games, realtime visualizations, virtual reality, and has recently started making its way in movie production—at least in previsualization and virtual production scenarios.

On the other hand, *physically based rendering* prioritizes the accuracy of the results. A very different set of techniques is employed to account for phenomena such as interreflections (light bleed), rough reflections, caustics, participating media and subsurface scattering, *e.g.* in human skin or food. These effects are realized through a simulation of the way light is emitted from light sources, travels through space, interacts with various objects in the scene, reaches an observer (camera) and finally, is recorded by a sensor. This simulation involves solving high-dimensional integrals, typically using *Monte Carlo* methods and appearance models that account for the interaction of light and matter.

Thanks to decades of research in this field, great progress has been achieved in the accuracy of the models, the amount of detail the images capture, and how efficiently they are computed. Realism has been a major driving force since the inception of the field of computer graphics. The fact that each step of the simulation is a relatively good approximation of the corresponding real-life phenomenon allows not only the creation of hyper-realistic images, but also broader industrial and scientific applications.

In this thesis however, we focus on the *inverse* problem: given an image, can we recover the state of the scene that lead to the formation of this image? Consider the task of determining the material or shape of an object from a set of images, illustrated in Figure 1.2. This is a surprisingly difficult problem, since the pixel values encode a complex superposition of indirect effects: for instance, a part of the object could appear brighter because a neighboring object reflects light towards it, or it could appear darker because another object casts a shadow. To solve this task, it is therefore not enough to focus on the object of interest alone–we need to understand its relationship to the surrounding environment as well. Correctly capturing this type of coupling is exactly what enables the kind of photorealistic rendering described above. Unfortunately, physically based

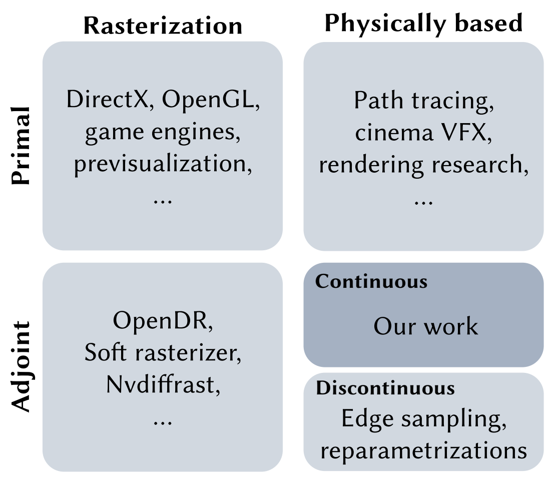

Rasterization
Physically based
Primal
Adjoint
DirectX, OpenGL, game engines, previsualization, …
Path tracing, cinema VFX, rendering research, …
OpenDR, Soft rasterizer, Nvdiffrast, …
Continuous
Our work
Discontinuous
Edge sampling, reparametrizations

Figure 1.1: Our work focuses on differentiable physically based rendering in order to accurately account for all aspects of light transport. While the algorithms proposed in this thesis do not account for gradients that arise due to visibility-related discontinuities, they can be extended to such cases [12].

rendering algorithms tend to be complex and fairly expensive–hours of rendering time for a single image are not uncommon.

We interpret a rendering algorithm as a function $f : \mathcal{X} \to \mathcal{Y}$, whose input encodes the shape and materials of objects. Evaluating $\mathbf{y} = f(\mathbf{x})$ yields a rendered image through the physical simulation of light transport. But the effects mentioned above, such as shadowing, specular interreflection, or subsurface scattering, obscure the individual properties of objects. Hence an inverse $f^{-1}(\mathbf{y})$ to recover scene parameters $\mathbf{x}$ from an image $\mathbf{y}$ is normally not available.

Instead, we focus on the efficient computation of *gradients* with respect to scene parameters. In other words, we will estimate the derivatives $\partial \mathbf{y}/\partial \mathbf{x}$ relating inputs and outputs of the simulation. Combined with a first-order optimization algorithm, we will then able to navigate the parameter space to find solutions to different types of inverse problems.

However, this is not a panacea: optimization objectives can be highly non-convex, and the observations used as reference often contain unavoidable ambiguities. For example, does an object appear to have a specific color and brightness because of the quality of the lighting, or the properties of its own surface? In practical applications, we will then have to pay significant attention to initial guesses and regularization to obtain satisfactory results.

We believe this pursuit to be highly pertinent, as many scientific applications rely on different kinds of optical measurements or image observations. Being able to reliably

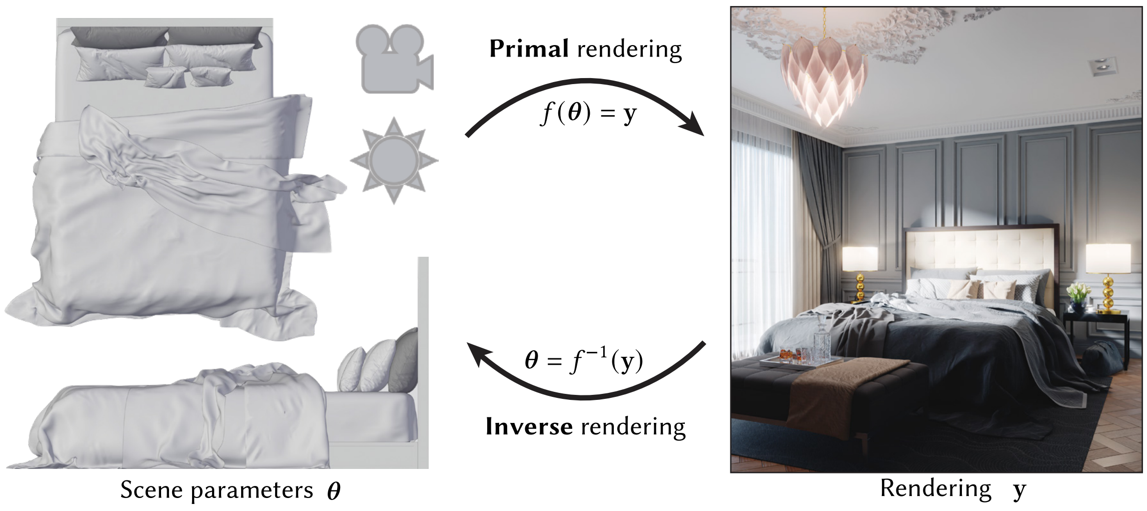

Primal rendering
$f(\boldsymbol{\theta}) = \mathbf{y}$
$\boldsymbol{\theta} = f^{-1}(\mathbf{y})$
Inverse rendering
Scene parameters $\boldsymbol{\theta}$
Rendering $\mathbf{y}$

Figure 1.2: Inverse rendering is the process of recovering the scene's properties from image observations. Since the rendering process is generally too complicated for any closed-form solution, we turn to gradient-based optimization. The work in this thesis does not rely on neural networks or other machine learning methods, but directly optimizes the parameter of physically based models. *Scene: "Bed Classic" by jiraniano on BlendSwap.*

invert the image formation process opens the door to solving challenging inverse problems ranging from relightable scene reconstruction to cloud tomography, non-light of sight imaging, computational lens design and 3D printing. This thesis contributes to this objective on three axes: physically based differentiable rendering algorithms, systems to efficiently realize those algorithms, and real-world applications leveraging them.

1.1 Overview

The topics covered in this thesis are summarized in Figure 1.3. We start by reviewing the relevant concepts and related work in Chapter 2.

Algorithms. In Part I, we discuss three methods to compute gradients with respect to scene parameters through physically based rendering. We first consider a method based on automatic differentiation, and find it to be limited by memory consumption. We then introduce *radiative backpropagation* and *differential ratio tracking*, two algorithms for unbiased and efficient differentiable rendering. The latter is dedicated to volumetric rendering and addresses bias and variance issues in existing methods.

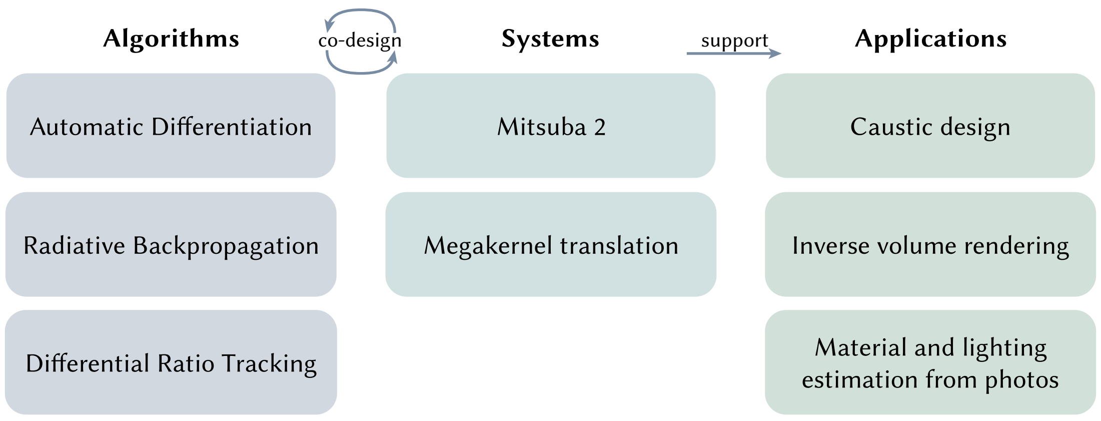

Algorithms
co-design
Systems
support
Applications
Automatic Differentiation
Radiative Backpropagation
Differential Ratio Tracking
Mitsuba 2
Megakernel translation
Caustic design
Inverse volume rendering
Material and lighting estimation from photos

Figure 1.3: This thesis covers contributions in the field of differentiable physically based rendering over three areas: algorithms, systems and applications.

Systems. Part II focuses on systems built for effective implementation and execution of differentiable rendering algorithms. We discuss the combination of constraints and challenges unique to physically based rendering. After reviewing the Enoki library [13], we present the architecture of Mitsuba 2, an open source *retargetable* renderer. It supports multiple computational backends (scalar CPU, SIMD on the CPU, GPU) and color modes (RGB, spectral, polarized). Importantly, it also supports automatic differentiation through the entire system. We then extend it with an automatic conversion to megakernel-style rendering for time- and memory-efficient implementation of the radiative backpropagation algorithm of Chapter 4.

Applications. Part III showcases three applications built from our algorithms and systems. Using Mitsuba 2's autodiff support, we optimize the heightfield of a slab of glass until it focuses light passing through into a desired caustic. Similarly, we optimize the spatially-varying index of refraction of a glass cube to project two separate caustic images. Next, we perform inverse volume rendering, recovering the density and albedo of high-resolution heterogeneous media from images. In the third applications, we simultaneously recover the lighting and material properties of real indoor scenes from pictures.

Finally, we review this thesis' contributions and turn to future research directions in Chapter 12. We believe differentiable rendering will prove to be a key tool in a wide range of applications, from computer vision and inverse rendering to broader scientific applications such as optics and climate science.

1.2 List of publications

The majority of this thesis is freely adapted from the following four published articles:

Chapters 7 and 9:
[14] **Mitsuba 2: A Retargetable Forward and Inverse Renderer**
Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Wenzel Jakob
ACM Trans. Graph. (Proc. SIGGRAPH Asia), Vol. 38, No. 6, pp 203:1–203:17, Nov. 2019.
`https://rgl.epfl.ch/publications/NimierDavidVicini2019Mitsuba2`

Chapters 4 and 8:
[15] **Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering**
Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, Wenzel Jakob
ACM Trans. Graph. (Proc. SIGGRAPH), Vol. 39, No. 146, pp 146:1–146:15, Jul. 2020.
`https://rgl.epfl.ch/publications/NimierDavid2020Radiative`

Chapter 11:
[16] **Material and Lighting Reconstruction for Complex Indoor Scenes with Texture-space Differentiable Rendering**
Merlin Nimier-David, Zhao Dong, Wenzel Jakob, Anton Kaplanyan
Eurographics Symposium on Rendering (DL-only Track), 2021.
`https://rgl.epfl.ch/publications/NimierDavid2021Material`

Chapters 5 and 10:
[17] **Unbiased Inverse Volume Rendering with Differential Trackers**
Merlin Nimier-David, Thomas Müller, Alexander Keller, Wenzel Jakob
ACM Trans. Graph. (Proc. SIGGRAPH), Vol. 41, No. 44, pp 44:1–44:20, Jul. 2022.
`https://rgl.epfl.ch/publications/NimierDavid2022Unbiased`

2 | Background

Physically based rendering (Section 2.2), and furthermore *differentiable* physically based rendering (Section 2.6) heavily rely on the numerical estimation of complex recursive integrals. We start by reviewing Monte Carlo integration, a crucial tool used to perform this estimation. We then turn to an introduction of physically based rendering, before reviewing automatic differentiation and other topics relevant to our work. Finally, we give an introduction to differentiable rendering in Section 2.6.

2.1 Monte Carlo integration

Unfortunately, the high-dimensional integrals describing light transport, which will be described in Section 2.2.2, are far too complex to compute analytically. Furthermore, deterministic integration methods such as quadratures will typically fall short as well, since their computational requirements typically grow exponentially with the dimensionality of the domain—which is known as the curse of dimensionality. Section 2.2 will discuss how this dimensionality quickly grows with the accuracy of the transport simulation.

Instead, *Monte Carlo* methods take a randomized approach: they draw samples from an (almost arbitrary) distribution, evaluate the integrand, and aggregate the sample's contributions. Under reasonable assumptions on the sampling distribution, this leads to an unbiased estimator that computes the solution of the integral *in expectation*, *i.e.* as the number of samples tends to infinity. A key advantage is that samples can be drawn with a cost linear with respect to the dimensionality of the domain. Furthermore, the independence of the samples makes Monte Carlo integration trivially parallelizable, which is crucial for the effective usage of modern hardware. In exchange, one has to accept variance, or noise, in the estimated result. Better sampling distributions, or simply more samples, can be used to lower the variance to an acceptable level.

2.1.1 Example problem

In the rest of this section, we will take the following toy problem as a guiding example. Over a bounded 2D domain $\mathcal{D}$, estimate the integral:

$$S = \int_{\mathcal{D}} f(\mathbf{x}) \ \mathrm{d}\mathbf{x}, \tag{2.1}$$

where f is a black-box binary function. It can be interpreted as computing the area of an unknown shape, visualized in Figure 2.2 (right), given a function testing whether a point is contained within the shape.

Let $\mathbf{x}$ be a random variable $\mathbf{x} \sim p(\mathbf{x})$, with p its *probability distribution function* (pdf). In the simplest cases, this will be a simple uniform distribution over the domain $\mathcal{D}$. The *cumulative distribution function* (CDF) of p is, by definition:

$$P(y) = \mathbb{P}(x \leq y) = \int_{-\infty}^{y} p(x)\ \mathrm{d}x. \tag{2.2}$$

2.1.2 Estimator

Monte Carlo integration transforms an integration problem into a stochastic sampling problem, *i.e.* how to best select $p(\mathbf{x})$ such that:

$$\mathbb{E}_p[\mathbf{x}] = S, \tag{2.3}$$

while keeping the variance of the estimate as low as possible. The Monte Carlo estimator of S is given by:

$$\langle S \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{p(\mathbf{x}_i)}, \tag{2.4}$$

where N is the sample count and $\mathbf{x}_i$ is the individual samples drawn independently from pdf. In our area estimation example, we can start with a simple uniform sampling strategy over the domain $[-1, 1]^2$, of area 4. All points are therefore sampled from the constant pdf $p(\mathbf{x}) = 1/4$, which produces the following easy-to-implement estimator:

$$\langle S \rangle = \frac{1}{4N} \sum_{i=1}^{N} f(\mathbf{x}_i), \tag{2.5}$$

The simplicity of this approach is a strength of Monte Carlo methods. This particular case, with $f(\mathbf{x}) \in \{0, 1\}$, is akin to *rejection sampling* or "dart throwing", which can be extremely inefficient. Real applications, as we will see below in the case of rendering, involve continuously-valued integrands with multiple factors, as well as sophisticated sampling techniques.

2.1.3 Importance sampling

There is flexibility with regard to the choice of sampling distribution $p(\mathbf{x})$. The only constraint is that it allocates a nonzero probability of sampling $\mathbf{x}$ for all values where the integrand is nonzero:

$$f(\mathbf{x}) \neq 0 \implies p(\mathbf{x}) > 0. \tag{2.6}$$

With *importance sampling*, p is set to be as close as possible to the integrand (up to a constant normalization factor). In fact, if it were possible to generate samples exactly proportionally to the integrand, the variance could be brought down to zero:

$$\begin{aligned} & p(\mathbf{x}) \propto f(\mathbf{x}) && (2.7)\\ \implies & \langle S \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{f(\mathbf{x}_i)/C}, && (2.8)\\ & = C, && (2.9) \end{aligned}$$

where C is the normalization constant of the pdf:

$$C = \int_{\mathcal{D}} p(\mathbf{x})\, \mathrm{d}\mathbf{x} = \int_{\mathcal{D}} f(\mathbf{x})\, \mathrm{d}\mathbf{x} = S, \tag{2.10}$$

which, of course, implies that Monte Carlo estimation is not needed if C is already known.

For most practical applications, this kind of perfect importance sampling will not be achievable for the entire integrand, but only certain subsets of its constituting factors. A crucial part of graphics research involves designing problem-specific sampling distributions to efficiently estimate integrals. Many sampling techniques have been proposed for standard rendering, but we will see they're highly relevant in the adjoint problem of gradient estimation as well [12, 15, 17].

2.1.4 Bias and variance

Estimators, as defined above, are computed by aggregating individual samples. In this thesis, we strive to build estimators that are: *unbiased*, *consistent*, and have low *variance*.

Variance. Variance quantifies the dispersion of estimators' values around their average. High variance values should be expected when one cannot design a sampling density that matches the integrand well enough for good importance sampling. In all practical applications, some amount of variance is to be expected. However, well-designed estimators will be able to minimize it. The consequence is that fewer samples will be needed to achieve the desired level of precision, bringing down the overall computational cost of the integration.

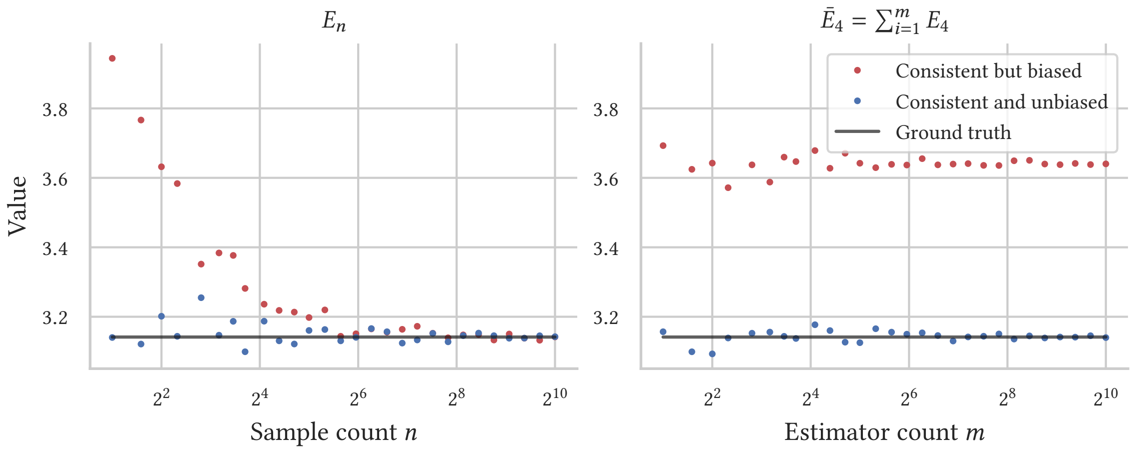


Figure 2.1: Left: *consistent* Monte Carlo estimators E_n using n random samples converge to the correct value as n increases. Regardless of its consistency, a *biased* estimator may be systematically over- or under- estimating the desired value (red dots). Right: taking averages of m biased estimators $\bar{E}_n$ for any finite n does not converge to the correct value (red dots, shown here with $n = 4$). On the other hand, a consistent *and* unbiased estimator converges as either n or m increase (blue dots).

While the asymptotic convergence rate of a Monte Carlo estimator remains $O\left(1/\sqrt{N}\right)$ [18], lowering the variance will improve the constant factor. The asymptotic rate itself can be improved by using quasi-random samples, see Section 2.1.7.

Consistency. An estimator is said to be *consistent* if, as the number of samples increases, the value of the estimator converges in expectation to the desired value. For example,

$$E_n = \frac{1}{n} \sum_{i=1}^{n} x_i + \frac{1}{n}, \tag{2.11}$$

where x_i are independent identically distributed random variables. E_n is consistent since

$$\lim_{n \to \infty} E_n = \mathbb{E}[x]. \tag{2.12}$$

However, it is *biased* because for any finite value n,

$$\mathbb{E}[E_n] = \mathbb{E}[x] + \frac{1}{n} \neq \mathbb{E}[x]. \tag{2.13}$$

Having unbiasedness *and* consistency implies that both estimators computed with large sample counts and averages of many instances of the estimators with a fixed sample count converge to the desired value. This is illustrated in Figure 2.1. In this example, the consistent-but-biased estimator approaches the desired value from above: the error distribution is not centered around zero.

Bias-variance tradeoff. For some applications, it may be too expensive or complex to build a correct, unbiased estimator. Likewise, there are cases where unbiasedness comes at the cost of significant additional variance. In those cases, a slightly biased, but lower variance estimator may actually achieve better overall convergence and therefore prove more practical.

2.1.5 Russian roulette

Consider an estimator for the following infinite sum:

$$G = \sum_{i=0}^{\infty} g(\mathbf{x}_i). \tag{2.14}$$

Infinite sums regularly appear in the context of physically-based rendering, where a light transport simulation should account for contributions of light sources after any number of interactions with the scene (Section 2.2.2). Of course, it is not possible to account for infinitely many terms, and the solution generally cannot be computed in closed form either.

Russian roulette (RR) can be employed to truncate the infinite sum while preserving an unbiased estimate of its complete value. Russian roulette truncates the sum after computing a random number of terms k, where the termination probability $1 - \mathrm{pdf}(k)$ can be chosen arbitrarily as long as $\forall\, k \geq 0,\ \mathrm{pdf}(k) > 0$. When the sum is *not* terminated at term k, all of the subsequent terms are weighted by $w = {}^{1}/_{\mathrm{pdf}(k)}$ to compensate for the eventual truncation. We denote p_i the probability of the sum reaching term i *and* not being terminated at i. Then,

$$\begin{aligned}\mathbb{E}[\hat{G}] &= p_0 \left[\frac{g(\mathbf{x}_0)}{p_0} + p_1 \frac{g(\mathbf{x}_1)}{p_0\, p_1} + (1 - p_1) \cdot 0 + \cdots \right] + (1 - p_0) \cdot 0 && (2.15)\\ &= \mathbb{E}[G]. && (2.16)\end{aligned}$$

In rendering, this technique is applied to turn infinitely recursive sampling processes into finite estimators without introducing bias.

2.1.6 Multiple importance sampling

As mentioned above, our integrals of interest will not admit perfect importance sampling schemes. In the rendering context, integrals will often consist of multiple terms that can be sampled individually but not jointly. In this situation, the collection of individual sampling techniques can be combined using *multiple importance sampling* (MIS) instead.

Essentially all state-of-the-art rendering techniques rely on MIS to achieve acceptable variance levels.

Suppose that techniques A and B produce samples $\mathbf{x}_A$ and $\mathbf{x}_B$ respectively. In order to benefit from the strength of both distributions, the key is to weigh the contributions of each sample by accounting for the probability of producing that sample using the other technique. Several ways to compute MIS weights have been proposed, with one of the most common choices being the *balance heuristic* [19]:

$$w(\mathbf{x}_A) = \frac{\mathrm{pdf}_A(\mathbf{x}_A)}{\mathrm{pdf}_A(\mathbf{x}_A) + \mathrm{pdf}_B(\mathbf{x}_B)}, \tag{2.17}$$

and vice-versa for sample $\mathbf{x}_B$'s weight. This can be generalized to more than two estimators and adapted to the characteristics of the problem, *e.g.* how well the individual estimators fit the integrands.

Later, Kondapaneni et al. [20] showed that this heuristic is not as optimal as it was once thought, and proposed new weighting functions that provably minimize the variance of the combined estimator—sometimes using negative weights.

2.1.7 Pseudo- and quasi-random numbers

Sampling strategies employed in Monte Carlo estimators often consume samples ξ uniformly distributed on the unit interval, and remap or *warp* them to the desired distribution $p(\mathbf{x})$. This transformation can be performed in various ways, from closed-form CDF inversion to hierarchical sample warping and rejection sampling.

We do not assume access to a true random number generator: high-quality pseudo-random numbers are sufficient for our purpose. In fact, the fully deterministic nature of PRNGs is precisely what allows *replaying* specific sequences of sampling decisions at will. This is a prerequisite of the *path replay backpropagation* algorithm of Vicini et al. [21], which we use in Chapter 5.

When running highly parallel simulations with hundreds of thousands of program instances drawing random numbers in parallel, it is important to ensure that the different instances are uncorrelated. Even algorithms that explicitly support large numbers of parallel streams, such as PCG [22], must be carefully seeded to avoid this issue.

Interestingly, the use of uniformly distributed pseudo-random variates as input to those transformations is not the optimal strategy: so-called *quasi Monte Carlo* (QMC) methods use carefully designed point sets that better cover the space by minimizing clumping. QMC and other stratification techniques [23] can *in theory* improve the convergence rate from $O\left({}^{1}/_{\sqrt{N}}\right)$ to $O\left({}^{\log(N)^k}/_{N}\right)$ [18]. In practice, however, the assumption

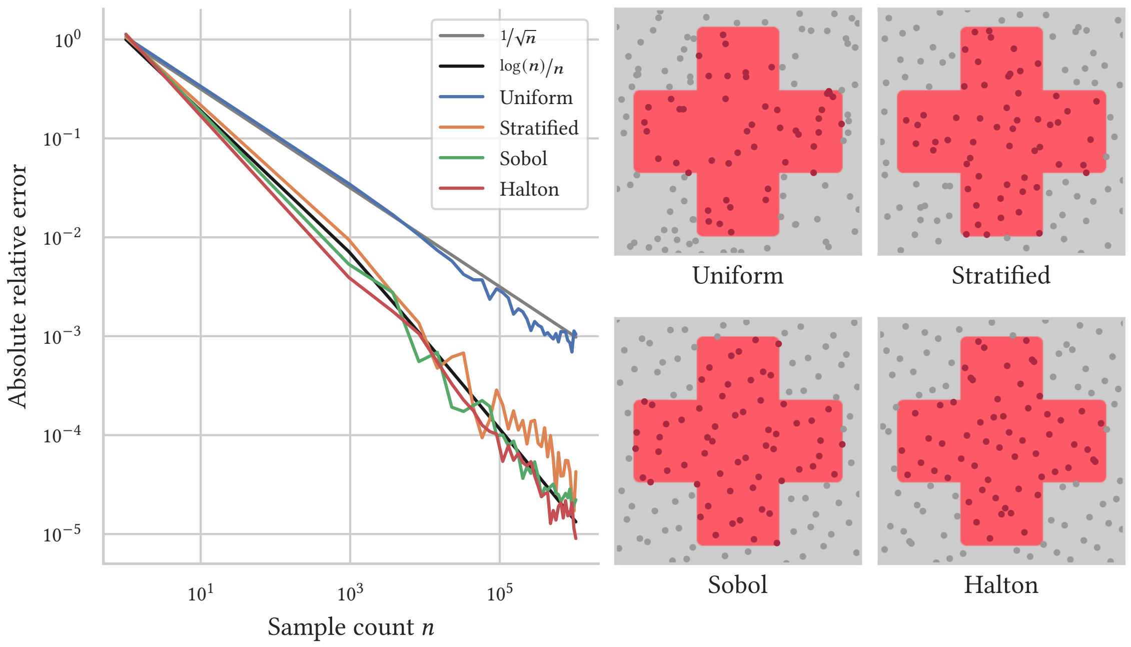


Figure 2.2: Estimating the area of the cross (red) with rejection sampling. Using the same Monte Carlo estimator with stratified random numbers, rather than purely uniform ones, converges faster to the desired value.

of a finite rate of variation of the integrand is not respected in the context of rendering. Still, an improvement of approximately 30% in the convergence rate is commonly observed. We illustrate the usage of quasi-random sequences for the simple rejection sampling-based problem of Equation (2.5) in Figure 2.2.

2.2 Physically based rendering

Physically based rendering (PBR) is concerned with producing images that faithfully represent the world by capturing, as accurately as possible, the physical processes involved in light transport. Many of the effects observed in our daily lives, such as the ones illustrated in Figure 2.3, require this kind of accurate simulation. Our goal is to learn about the real world from image-based observations: it makes intuitive sense that one would gain more useful information from differentiating and inverting a physically based rendering process rather than rasterization. This thesis therefore focuses exclusively on physically based methods.

As we will see, geometric light transport can be expressed as a Fredholm integral equation of the second kind, which can then be estimated with Monte Carlo integration. This section only introduces the background relevant to the work presented in this the-

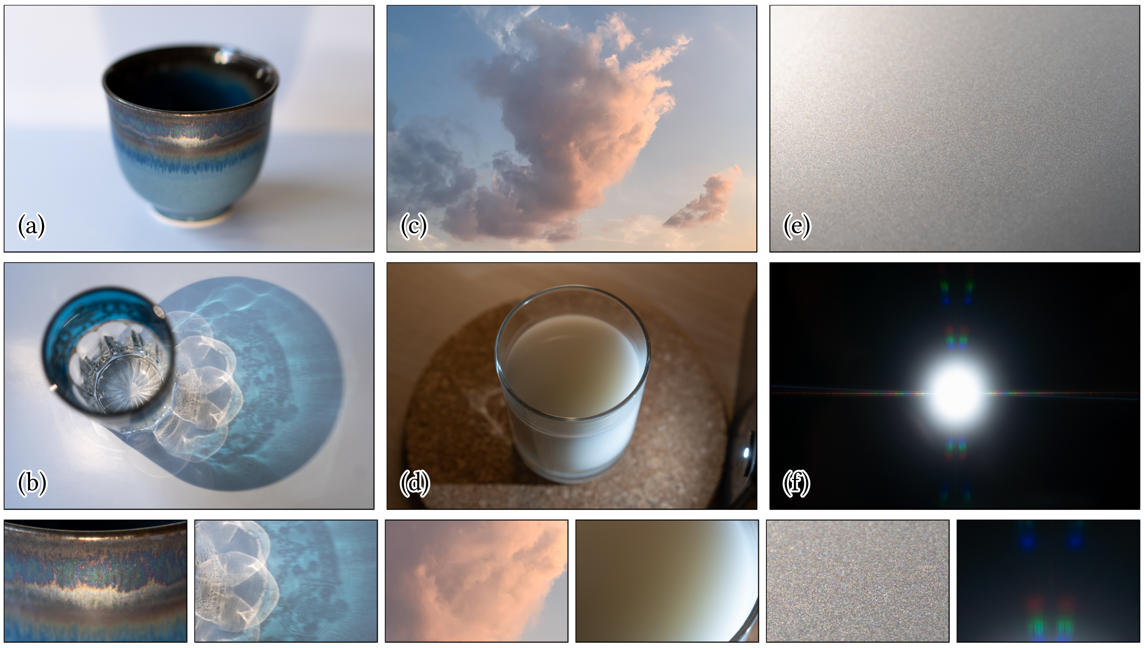


Figure 2.3: Physically based rendering attempts to faithfully simulate how light interacts with the world. This includes **(a)** macro-level spatially varying surface reflectance properties, **(b)** reflective and refractive caustics, **(c-d)** internal scattering and absorption within participating media, **(e-f)** micron-scale structures leading to wave effects, as well as other characteristics such as polarization. We will differentiate the rendering process in the presence of **(a-d)**, and leave advanced effects **(e-f)** to future work.

sis. For a complete treatment, please see the reference textbook of Pharr et al. [24] as well as the thesis of Veach [25].

2.2.1 Setting and scope

Since we must eventually be able to invert the light transport simulation, we will have to restrict its scope and accuracy to the phenomena that are relevant to the type of scenes and objects that we would like to analyze. In most cases, the algorithms used can be adapted or generalized on a per-application basis to support the relevant effects. For example, we will not attempt to model wave effects and diffraction because they have a limited impact on the applications pursued in this thesis.

Scene model. We will simulate light transport involving geometric optics. The scenes will be made up of the following components, following the well-established abstractions of PBRT [24] and popular rendering frameworks [26]:

- **Emitters** produce light by emitting radiance into the scene (*e.g.* the sun or the filament of a lightbulb). They are characterized by their position, intensity and angular profile.

- **Participating media** model regions where microscopic particles can absorb, scatter and emit light (see Section 2.2.3).
- **Shapes** define the specific position and orientation of the surface boundaries of objects. Common representations include triangle meshes, analytic shapes, and signed distance functions.
- **Surface characteristics**, such as their roughness and color, are captured by the *bidirectional reflectance function* (BSDF), which will be introduced in Section 2.2.2.
- **Sensors** receive and record incident radiance hitting their surface. The most common sensors used are modeled after cameras, but they can represent any kind of light-based measurement device.

Light properties. Light is an electromagnetic radiation characterized by its wavelength, which is perceived as its color. Correct handling of color in rendering involves representing surface, emitter and sensor properties such as their reflectance, intensity and sensitivity as a function of the wavelength. Moreover, one must consistently use input and output color spaces and image storage formats that can represent a large enough set of colors (gamut). Methods have been proposed to smoothly upsample RGB reflectance data to spectral distributions [27, 28, 29].

Beyond its wavelength, light is also characterized by its *polarization* state, which is the direction of the electric field. Light's polarization can be affected by interactions with surfaces, which provides additional clues about the observed objects' materials and surface geometry.

Spectral and polarization properties are important for the accurate simulation of light [30, 31]. Accurate measurements could help reduce the many degrees of freedom of our challenging and underconstrained reconstruction problems [32, 33]. However, it also places additional requirements on the acquisition setup, and takes it out of the reach of casual users (*e.g.* consumer cameras do not output a full spectral distribution or the polarization state of light). Therefore, our experiments will be limited to unpolarized light simulated in an RGB color basis. Nevertheless, the algorithms presented are agnostic to the color representation and incorporating these aspects is an important avenue for future work.

2.2.2 Rendering equations for the surface case

We now describe the fundamental equations underpinning light transport, following the exposition of Veach [25, Section 3.7].

The measurement equation. In general terms, physically based rendering outputs measurements of scalar-valued radiometric quantities $I_1, \ldots, I_M$—for example, the individual pixels of an M-pixel image. The *measurement equation* [25] describes the role of the sensor in forming those measurements:

$$I = \int_{\mathcal{M}\times\mathcal{S}^2} W_e(\mathbf{x}, \boldsymbol{\omega})\, L_i(\mathbf{x}, \boldsymbol{\omega})\ \mathrm{d}A(\mathbf{x})\, \mathrm{d}\Omega^{\perp}_{\mathbf{x}}(\boldsymbol{\omega}), \tag{2.18}$$

where $\mathcal{M}$ is the sensor surface, $\mathcal{S}^2$ is the space of directions on the unit sphere, W_e is the sensors' responsivity at position $\mathbf{x}$ and incident direction $\boldsymbol{\omega}$, L_i is the incident radiance[1], $\mathrm{d}A$ is the area measure on the sensor's surface, and $\mathrm{d}\Omega^{\perp}_{\mathbf{x}}$ is the projected solid angle measure at $\mathbf{x}$. The projected measure induces a *foreshortening factor* $|\cos\theta_i|$, where θ_i is the angle between the incident direction and the surface normal.

The light transport equation. Next, the *light transport equation* [34] describes the relationship between the incident radiance L_i at any point $\mathbf{x}$ and incident direction $\boldsymbol{\omega}$, and the emitted and in-scattered radiance. It can be derived from the *radiative transfer equation* and the principle of conservation of power [35].

Here, we use the angular form following Veach [25, Equation (3.19)], but equivalent surface and operator forms exist (Chapter 4 will rely on the latter). We assume that the radiance distribution in the scene has reached its steady state—though time-resolved rendering has important applications for inverse reconstructions such as non-line-of-sight imaging [36]. Furthermore, we assume for the moment that the scene does not contain participating media. Light therefore travels in a straight line:

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = L_o(\mathbf{r}(\mathbf{x}, \boldsymbol{\omega}), -\boldsymbol{\omega}), \tag{2.19}$$

where $\mathbf{r}$ is the *ray casting function*, which returns the closest intersection point to the ray with origin $\mathbf{x}$ and direction $\boldsymbol{\omega}$. The exitant radiance L_o is computed as

$$L_o(\mathbf{x}, \boldsymbol{\omega}_o) = L_e(\mathbf{x}, \boldsymbol{\omega}_o) + \int_{\mathcal{S}^2} f_s(\mathbf{x}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_o)\, L_o(\mathbf{r}(\mathbf{x}, \boldsymbol{\omega}_i), -\boldsymbol{\omega}_i)\ \mathrm{d}\Omega^{\perp}_{\mathbf{x}}(\boldsymbol{\omega}_i), \tag{2.20}$$

[1]For more details on the radiometric quantities and units involved, see PBRT [24, Chapter 5].

where L_e is the radiance emitted at the current position $\mathbf{x}$ towards the direction of observation $\boldsymbol{\omega}_o$. The integral, taken over the sphere of directions $\mathcal{S}^2$, computes radiance reaching point $\mathbf{x}$ from all directions $\boldsymbol{\omega}_i$ and reflected in direction $\boldsymbol{\omega}_o$. Finally, f is the *bidirectional scattering function* (BSDF), which models the proportion of radiance reflected or transmitted by the current surface from $\boldsymbol{\omega}_i$ to $\boldsymbol{\omega}_o$.

Note that Equation (2.20) is self-referential: L_o appears on both the left- and right-hand sides, which lends itself to recursive computational algorithms. We will rely on Monte Carlo integration to estimate it efficiently despite its potentially infinite dimensional integration domain.

2.2.3 Participating media

The surface case of the rendering equation described above has been generalized to account for the effect of microscopic absorbing, scattering, and emissive particles encountered along light paths. Examples of participating media include clouds, fire, smoke, foods, milk and human skin (Figure 2.3).

Medium properties and notation. We consider a heterogeneous medium with small, spherical, independently distributed, scattering and absorbing particles. Such a medium is described for all positions $\mathbf{x}$ by its spatially varying *absorption* $\sigma_a(\mathbf{x})$ and *scattering* coefficients $\sigma_s(\mathbf{x})$ which are proportional to the density of particles along a ray (units of m^{-1}). The *extinction* coefficient $\sigma_t := \sigma_a + \sigma_s$ is defined as the sum of absorption and scattering.

In practice, it can be useful to parametrize media by $\boldsymbol{\theta} := (\sigma_t, \alpha)$, where the *scattering albedo* $\alpha := {}^{\sigma_s}/_{\sigma_t}$ captures the probability of scattering (as opposed to absorption) at an interaction with the medium. When referring to a position $\mathbf{x}_t = \mathbf{x}_0 + t \cdot \boldsymbol{\omega}$ along a ray $(\mathbf{x}_0, \boldsymbol{\omega})$, we will use t and $\mathbf{x}_t$ interchangeably to avoid cluttering formulae. The *transmittance* along the ray models the proportion of light traveling between points a and b without being absorbed. It is defined as

$$\mathrm{T}(a, b) := \exp\left(-\int_a^b \sigma_t(s)\ \mathrm{d}s\right), \tag{2.21}$$

where $\sigma_t(s) \equiv \sigma_t(\mathbf{x}_0 + s \cdot \boldsymbol{\omega})$. We use the shorthand $\mathrm{T}(t) \equiv \mathrm{T}(0, t) \equiv \mathrm{T}(\mathbf{x}_0, \mathbf{x}_t)$.

Volume rendering equation. Volumetric light transport is governed by the volume rendering equation [35, 37, 38], an integrodifferential equation modeling the absorbed, emitted, in-scattered and out-scattered radiance. We use the pure integral form of the volume rendering equation in the following, as it lends itself better to estimation with Monte Carlo methods.

$$\begin{aligned} L_o(\mathbf{x}, \boldsymbol{\omega}_o) = \int_0^{t_s} \mathrm{T}(t) \left[\sigma_a(t)\, L_e(t) + \sigma_s(t)\, L_s(t, \boldsymbol{\omega}_o)\right] \mathrm{d}t \\ + \mathrm{T}(t_s) \left[L_e(t_s) + L_s(t_s, \boldsymbol{\omega}_o)\right], \end{aligned} \tag{2.22}$$

where $L_i(\mathbf{x}, \boldsymbol{\omega})$ is the radiance in direction $\boldsymbol{\omega}$ at position $\mathbf{x}$ and L_e denotes emission from the medium, a surface, or from the environment. t_s corresponds to the distance along the ray to the closest surface or medium boundary. L_s is the in-scattered radiance on volumes or surfaces:

$$L_s(\mathbf{x}, \boldsymbol{\omega}_o) := \int_{S^2} L_o(\mathbf{x}, \boldsymbol{\omega}_i)\, f_s(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i)\, \mathrm{d}\boldsymbol{\omega}_i\,, \tag{2.23}$$

where f_s denotes either the *phase function* or the bidirectional scattering distribution function (BSDF) depending on whether $\mathbf{x}$ lies on a surface or in a medium.

Extensions. Further generalizations of Equation (2.22) have been proposed to capture a broader class of participating media. For instance, the shape of the individual particles composing the medium may have varied profiles, introducing anisotropic behavior. This was first studied in other fields [39, 40, 41], with micro-flake theory [42] being proposed in graphics to capture this effect. Moreover, the spatial distribution of the particles may not be uniform but correlated, leading to a theory and algorithms dedicated to the simulation of non-exponential media [43, 44, 45, 46].

2.2.4 Rendering algorithms

Physically based rendering relies on dedicated Monte Carlo integration algorithms to estimate the high-dimensional Fredholm integral equations (2.20) (surface case) and (2.22) (volume case).

Dimensionality The path integral measurement equation of Veach [25] gives a better understanding of the problem's dimensionality:

$$I_j = \int_{\Omega} f_j(\bar{x})\ \mathrm{d}\mu(\bar{x})\,, \tag{2.24}$$

where Ω is the set of all transport paths of all lengths, $\bar{x}$ is a path in Ω, and f_j is the measurement contribution function. The dimensionality of the integration domain Ω is infinite.

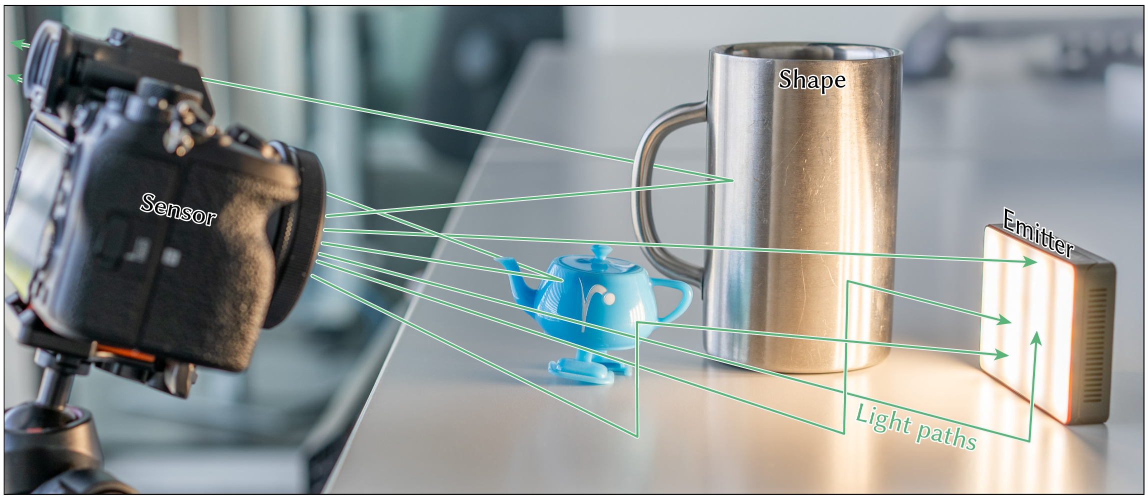


Figure 2.4: The path tracing algorithm simulates the travel of light rays through a scene *in reverse* using geometric optics. Paths start from the camera (left) and travel in a straight line until they hit the closest surface. A new direction is sampled according to the surface properties (center) and the path throughput is updated to account for the surface's color and reflectance. When a light source is encountered (right), the emitted radiance is added to the result. When rays eventually escape the scene bounds, they are terminated and their contribution is added to the image.

Deterministic numerical integration algorithms such as quadratures are therefore not viable due to the *curse of dimensionality*: the amount of computation needed grows exponentially with the dimensionality of the integration domain. On the other hand, Monte Carlo methods for rendering enjoy linear time complexity with respect to the number of interactions along light paths, which makes it a particularly appropriate tool. Techniques such as Russian roulette (Section 2.1.5) are employed to ensure that the simulation completes in finite time.

Path tracing. Within the framework of Monte Carlo integration, the efficiency of the estimators will largely depend on the quality of the distributions used for importance sampling (Section 2.1.3). In other words, one must strive to sample the integration variables of Equation (2.20), such as $\boldsymbol{\omega}_i$, using a distribution matching the integrand as closely as possible.

The *path tracing* algorithm iteratively constructs light paths, where each path segment corresponds to one level of recursion in the light transport equation. Its simplest form is visualized in Figure 2.4. Light paths are built in reverse compared to light's true direction of travel: they start from the sensor rather than light sources. This reversal is possible due to the reciprocity of the physical equations, a property we will leverage as well in Chapter 4. Sampling efficiency is increased in many cases, as samples are guaranteed to reach the sensor.

The algorithm starts by sampling a position $\mathbf{x}$ on the sensor's surface and direction $\boldsymbol{\omega}$ leaving the sensor (Equation (2.18)). In the case of an idealized pinhole camera, $\boldsymbol{\omega}$ is uniquely determined by $\mathbf{x}$, while a more realistic camera model will sample $\boldsymbol{\omega}$ based on the aperture size. Then, the path contribution is initialized to zero and a recursive incident radiance estimation is performed. The corresponding pseudocode is given in Listing 1.

```
def path_trace(scene, sensor):
    # Sample an initial direction leaving the sensor
    ray = sensor.sample()
    L = 0  # Path radiance
    β = 1  # Path throughput
    while True:
        # Terminate infinite sum with Russian roulette
        q = russian_roulette(throughput)
        if random() < (1 - q):
            break
        β /= q

        # Find the closest surface interaction in the direction of the ray
        si = scene.intersect(ray)
        # Account for emission at that point and in our direction
        L += β * si.emitter().eval(ray)

        # Importance sample the next ray from the surface's BSDF
        ray, weight = si.bsdf().sample(ray)
        # Among other things, this sampling weight accounts for the surface color
        β *= weight

    # Finished, return accumulated radiance
    return L
```

Listing 1: Simplified pseudocode for the path tracing algorithm without next-event estimation.

Next-event estimation. *Next-event estimation* (NEE) improves sample efficiency by additionally attempting a direct connection to light sources at each interaction [47]. Sampling a direction toward a light source proportionally to its contribution is an interesting problem in itself, especially in scenes containing many light sources or with heavy occlusion. The contribution from NEE is combined with the main path's contribution using multiple importance sampling (MIS, Section 2.1.6).

Volumetric path tracing. Path tracing has been extended to account for participating media [48], which can absorb or scatter light over entire regions of space rather than at discrete surfaces. Once entering a medium's boundary, volumetric path tracing repeatedly samples *free-flight distances* based on the density of the medium. They correspond to the distance traveled by a light ray within the medium before interacting with one of the medium's particles and being scattered. The scattering direction is sampled from the *phase function*, which is a generalization of the BSDF for the volume case. Between scattering events, the throughput of the path is reduced by the medium's absorption. When next-event estimation is used, one must take care to account for the attenuation along the direct emitter connection.

Extensions. A wealth of research is dedicated to improving the algorithms above, whether to improve their efficiency or account for more advanced phenomena. For example,

- *Particle tracing*, also called light tracing, simulates paths starting from light sources rather than the sensor. It is can perform better than path tracing in certain situations, such as the caustic design scenario of Chapter 9.
- *Bidirectional path tracing* [49] simulates paths starting from *both* sensors and light sources, and then explicitly connecting them.
- *Photon mapping* [50, 51] and *virtual point light* [52] methods are typically biased, but can significantly reduce variance in cases where the above methods may not be able to create connections to either emitters or sensors.
- *Manifold exploration* [53] and *manifold next-event estimation* [54, 55] can handle paths containing series of specular interactions, that are hard or impossible to sample with existing methods. Such paths occur for example in human eyes.
- *Time-resolve* or *transient* path tracing [56] can produce images integrated over extremely short time periods, accounting for the time of travel of photons. Longer light paths therefore only contribute to the later time slices.

Chapter 4 shows that estimating scene parameter gradients can be recast as an integration problem, for which we use an algorithm similar to (volumetric) path tracing. Finally, we examine the volumetric case in more detail in Chapter 5 and propose a sampling strategy dedicated to the estimation of gradients with respect to the medium properties.

2.3 Automatic differentiation

Our goal is to compute scene parameter gradients through the rendering process in order to invert it. While it is certainly possible to manually derive and implement the derivative computation for each function involved [57], this is tedious, error-prone, and leads to implementations that are hard to extend. *Automatic differentiation* (AD) systems track the differential relationship between variables in a program. Given the complexity of physically based rendering algorithms and the number of components involved, AD is an enticing solution to the challenges with manual differentiation mentioned above.

Chapters 3 and 7 discuss the algorithmic and systems implications of applying AD to differentiable physically based rendering. The fundamentals are introduced here, but we refer the reader to the reference book of Griewank and Walther [58] for a more thorough treatment. *Forward* and *reverse* mode differentiation are two ways to transform a computation (the *primal*) into programs that evaluate corresponding derivatives. Note that this distinction is relevant whether the transformation is carried out automatically or manually. In the following, we briefly contrast forward- and reverse-mode differentiation of a simple example function

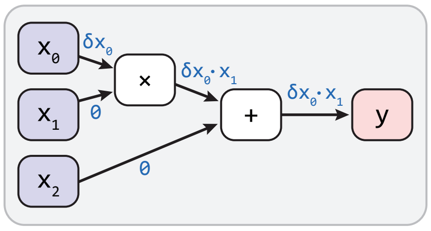
$$f(x_0, x_1, x_2) \coloneqq x_0 \cdot x_1 + x_2. \tag{2.25}$$

We denote $\mathbf{J}_f$ the Jacobian matrix

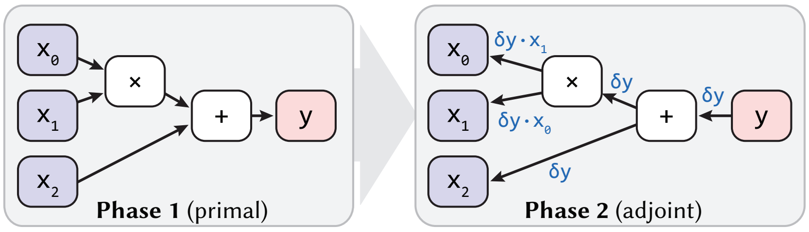
$$\mathbf{J}_f \coloneqq \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}, \tag{2.26}$$

where n and m are the dimensionalities of the function's inputs and outputs respectively.

Forward-mode. Forward-mode propagates derivatives from inputs to outputs by repeated application of the chain rule. Below, it propagates a perturbation δx_0 along edges, resulting in the partial derivative ${}^{\partial f}\!/\!{}_{\partial x_0} = x_1$. More generally, it can evaluate arbitrary directional derivatives $\mathbf{J}_f\, \boldsymbol{\delta}_{\mathbf{x}}$ using a single pass. However, separate derivatives with respect to individual inputs require multiple passes and are therefore costly, making forward-mode inadequate for most of our applications of interest. Nevertheless, it is comparatively simpler to implement and remains a useful tool to visualize or validate the effect of individual inputs on the output.



Reverse-mode. Also called *backpropagation*, reverse-mode differentiation evaluates the chain rule in reverse, from outputs to inputs, and is ideal for optimization of functions with one output and many inputs. Below, it computes grad $f(\mathbf{x}) = (x_1, x_0, 1)$.



When f is multi-valued, reverse-mode can evaluate arbitrary directional derivatives $\mathbf{J}_f^T\, \boldsymbol{\delta}_\mathbf{y}$ with respect to perturbations of the outputs $\boldsymbol{\delta}_\mathbf{y}$ in a single pass. The program execution order must be reversed during the propagation of derivatives. When reverse-mode derivation is performed automatically, this is generally accomplished by recording a transcript of operations (also known *Wengert tape* [59] or *computation graph*) during the primal phase that is replayed in a subsequent adjoint phase. As part of this process, each primal variable is associated with a corresponding *adjoint variable* (*e.g.* $\delta x_0 = \delta y \cdot x_1$) that tracks its sensitivity with regard to perturbations of the output. Because we are interested in optimizing parameters of scenes including potentially large textures, resulting in millions of input parameters, we focus exclusively on reverse-mode AD in the following.

Usage of AD. *Backpropagation* [60] is a key ingredient of deep learning. In recent years, libraries such as TensorFlow [61], PyTorch [62] and Jax [63] powered the development of deep learning and significantly lowered the barriers to entry. These libraries offer a high-level interface manipulating tensor and neural network primitives, while gradient computation is transparently handled by an AD backend. Undoubtedly, the ease of use and flexibility afforded by AD contributed to the fast pace of research in this field.

Unfortunately, these systems are designed and optimized for relatively structured and regular computational workloads, with high arithmetic intensity operations such as matrix-matrix multiplication and convolutions. Neural networks, as well as their gradient computations, typically decompose into such operations, for which hardware vendors have developed highly-tuned kernels. Machine learning libraries, then, must combine these kernels in the right order to realize the higher-level operations specified by the user—as well as their adjoints.

The physically based rendering algorithms, however, inherently come with a higher degree of dynamism: scenes loaded at runtime may contain any combination of components (shapes, BSDFs, emitters, etc), each with their specific code. Furthermore, global illumination implies that light paths may encounter any combination of scene elements in any order, adding to the irregularity of the workload. These operations cannot be realized with a simple combination of the aforementioned optimized kernels. This imposes heavy design constraints on systems tackling this problem, which are discussed in Chapter 6.

Runtime cost. Maintenance and traversal of the reverse-mode transcript add considerable additional runtime cost, which can be amortized by differentiating multiple coherent evaluations of f at the same time—Griewank and Walther [58] refer to this as *vector mode*.

For long-running computations, the memory requirements of reverse-mode differentiation are prohibitive. Checkpointing strategies [64] reduce storage overheads by discarding information that can be recovered later on by repeating parts of the computation, but this introduces considerable additional complexity. Furthermore, the size of the checkpoints themselves can be problematic in vector-mode differentiable renderers, which normally propagate large wavefronts consisting of many millions of rays.

Chapters 3 and 7 investigate the usage of AD for differentiable physically based rendering, including the associated tradeoffs.

2.4 Adjoint methods

In some cases, there exists a third alternative to either manual analytic gradient derivation or automatic differentiation: the *adjoint sensitivity method.* It has been applied to optimal control [65, 66], fluid simulation [67], retargeting the motion to elastic robots while minimizing vibrations [68], and to reduce the cost of training neural residual networks [69] in the area of machine learning. The tutorial of Bradley [70] provides a good introduction to the topic.

Commonly used in time-dependent partial differential equations (PDEs), the adjoint method enables efficiently estimating gradients with respect to the input parameters (*e.g.* the system's initialization) by simulating an adjoint equation *backward in time*. In such cases, once the adjoint equation has been derived for the target problem, the reverse simulation typically has time and memory complexities comparable to the forward simulation.

In Chapter 4, we propose a differentiable rendering algorithm resembling the adjoint method in that it also creates an adjoint problem that can be sampled in reverse. However, the specifics of the two approaches are different because the steady state of light transport lacks a natural time variable that we would simulate in reverse. Replacing the bulk of the transcript by an adjoint transport algorithm makes it possible to apply AD *symbolically* and only where needed, during a pre-processing step. The transcript is no longer required, drastically reducing memory usage. A system implementing this approach is presented in Chapter 8.

2.5 Compilers and domain-specific languages

Compilers. Compilers are programs that transform (generally higher-level) source code to another (generally lower-level) representation. Compilers are often split into *frontends* that transform source language code to an *intermediate representation* (IR), and *backends* that transform the IR into platform-specific binaries or instructions. A number of optimizations passes are applied at each representation level in order to produce an efficient program, while maintaining the high-level constraints of the programming model (*e.g.* correctness of arithmetic operations and memory integrity). LLVM [2] is a highly successful compiler toolchain that follows this architecture.

The ISPC compiler [71], also based on LLVM, transforms source code written in a slightly modified version of the C programming language into efficient *vectorized* machine code. In this context, *vectorization* refers to the use of *single instruction / multiple data* (SIMD) instructions, which perform the same operation on 2, 4, or 8 operands at once. It has been used successfully in high-performance computing and graphics applications.

Just-in-time compilation. *Just-in-time* (JIT) compilers perform the same task, but at runtime rather than in an offline pre-processing phase. Additional optimization opportunities thus become available: for example, in rendering, the contents of the scene to render are known. This information makes it possible to compile only the required components of the system, and *e.g.* combine or fuse them. Or, in other contexts, the input data and statistics on the most used parts of the program (hot paths) can be leveraged to apply further optimizations. On the other hand, the execution time of JIT compilation must be tightly controlled as it takes place at runtime.

DSLs for graphics. Finally, *domain-specific languages* are languages whose features and syntax are designed with a specific target domain in mind, as opposed to general languages such as C. DSLs can be *embedded* in another, more general language, taking the form of a library with potential overloading of the host languages' syntax and operators.

A number of domain-specific languages (DSLs) have been proposed to accelerate the development of efficient numerical algorithms in the area of computer graphics. Halide [72] facilitates the design of highly optimized image processing pipelines, decoupling the computation from the way it is carried out to expose optimization opportunities. The approach can be combined with reverse-mode AD to create new neural network layers or solve inverse problems [73].

Anderson et al. [74] recently proposed *Aether*, a DSL for Monte Carlo rendering which symbolically differentiates sampling code to determine associated probability densities that are crucial for many rendering techniques. *Rodent* is a related system by Pérard-Gayot et al. [75], which builds on *AnyDSL* [76] to generate a vectorized implementation for rendering a specific input scene, while applying partial evaluation to combine and specialize the individual components of a renderer.

For inverse rendering, Yang et al. [77] propose a language and compiler that extends automatic differentiation to both continuous *and* discontinuous functions. Their method outputs TensorFlow, PyTorch, Halide or GLSL programs and is evaluated on procedural shaders. Bangaru et al. [78] likewise introduce a language dedicated to the evaluation of derivatives at parametric discontinuities. Integration and the Diract delta are both introduced as language primitives for explicit handling. The resulting system, Teg, is evaluated on various inverse tasks from procedural shader parameter optimization to trajectory optimization in a minigolf physics simulation with contacts.

Our work on Mitsuba 2 (Chapter 7) is related to Rodent, but targets a wider set of transformations including changes to the formulation of light transport (*e.g.* polarization) and problem statement (*e.g.* inverse rendering via differentiation) but does not specifically focus on partial evaluation—although combining both approaches is likely feasible. Mitsuba 2 and Mitsuba 3 [79] are built onto Enoki [13] and Dr.Jit [80] respectively, which can be seen as embedded DSLs and include an optimizing JIT compiler. The latter also allows targeting different execution models and platforms, such as GPUs or vectorized CPU instructions, at runtime.

2.6 Differentiable rendering

We now introduce the main topic of this thesis: differentiable rendering. Fundamentally, we are concerned with the inversion of the physically based rendering problem of Section 2.2. Although research in inverse graphics dates back several decades, great strides have been taken over the last few years. While this thesis is focused on the graphics approach to the problem, highly related inverse problems have been studied in neutron transport and computer vision. We discuss some of these efforts below.

2.6.1 Problem statement

Inverse rendering seeks to invert the rendering process to infer the properties of the object or phenomenon that was observed. This inversion could be carried out in different ways, but the physically based rendering process is generally far too complex to admit closed-form solutions. Instead, we turn to iterative gradient-based methods.

Differentiable rendering, then, is concerned with the computation of the required parameter gradients. This thesis presents algorithms and systems dedicated to their efficient and correct estimation.

Note that while differentiable rendering is of course highly useful for inversion problems, local gradients can also be leveraged in other ways, such as improving the efficiency of forward rendering [55, 81, 82].

2.6.2 Neutron transport

The earliest related work involving derivatives of Monte Carlo simulation can be traced back to the field of neutron transport [83, 84]. Indeed, simulations used in the design of nuclear reactors involve much of the same theory and algorithms, as light transport.

Monte Carlo perturbation analysis methods were developed to study the effect of small parameter changes in nuclear reactors on *e.g.* reaction rates. Rief et al. [83] classify them in three categories: *correlated tracking*, *derivative operator sampling*, and *use of importance.* In all cases, the main goal is to estimate the difference in outcome between two configurations, despite the variance of each simulation's outcome. Several variance reduction schemes commonly deployed in rendering today, such as Russian roulette and splitting, were already in use. Correlated tracking forces the modified configuration to use the same sampling decisions as the base configuration, but re-evaluates the sampling weights. On the other hand, derivative operator sampling estimates the value of the difference using a Taylor expansion, which implies the estimation of parametric derivatives.

Rather than the comparison of two configurations, our focus on derivatives is motivated by the application of iterative gradient-based optimization in order to recover scene parameters from observations. This is highly related to *e.g.* the work of Hall [85], who proposed applying a Newton-Raphson iteration to optimize the experimental parameters with respect to a log-likelihood objective. Derivatives were also used for *sensitivity analysis* [84, Chapter 6.II], for example to determine the effect of the uncertainties in the reactors' characteristic parameters on the reaction rates.

2.6.3 Differentiable rasterization

Beginning with the work on *OpenDR* by Loper and Black [86], a number of approximate differentiable rendering techniques have been proposed in the field of computer vision. Relying on smooth rasterization of meshes or volumes [87, 88, 89, 90], these methods focus on primary visibility without accounting for indirect effects (shadows, interreflection, etc.). Petersen et al. [91] generalized smooth rasterization methods and studied the effect of different aggregation functions.

Because it can rely on the hardware-accelerated standard graphics pipeline on GPUs, differentiable rasterization can achieve impressive performance. Laine et al. [92] report run times of 4.5 to 6.5 ms per 2048×2048 frame, including both the primal and gradient estimation passes. Their analytic anti-aliasing technique is illustrated in Figure 2.5. This level of performance is particularly beneficial when incorporating a differentiable rendering component within a larger end-to-end differentiable pipeline.

As an alternative to smooth rasterization, Zhou et al. [93] proposed differentiable *vectorization.* Here, "vectorization" should be taken in the sense of *vector graphics*: instead of rasterizing triangles to a discrete bitmap, the individual triangles are drawn to an infinite-precision canvas. The final pixel color is thus smoothly related to the vertex positions. This yields noise-free direct illumination, as well as noise-free gradients—

although limited to direct illumination, and with a cost expected to rise with the number of visible edges. In the 2D domain, Li et al. [94] propose a differentiable vector graphics rendering pipeline.

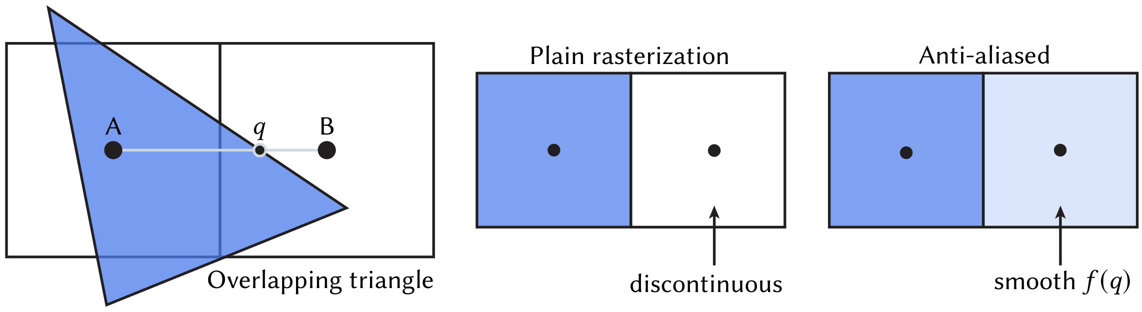


Figure 2.5: In the most basic form of rasterization, triangles from the scene are projected to the image plane (left). After accounting for the triangles' distance to the camera to resolve occlusion, a pixel takes the color of a triangle if it overlaps with its center (center). However, this results in hard color transitions and aliasing, which is particularly visible in diagonal lines. Moreover, this discrete choice introduces a discontinuity with respect to the triangle vertex coordinates, which is not differentiable. Laine et al. [92] compute the color of adjacent pixels as a function of the surface overlap, which is itself a function of the vertex coordinates (right). A continuous relationship is therefore established between the scene parameters and the final pixel color, allowing for differentiation. Other forms of smooth rasterization operate similarly. Figure adapted from [92, Figure 3].

2.6.4 Physically based differentiable rendering

At the other end of the spectrum, physically based differentiable rendering methods account for global illumination and more advanced phenomena such as participating media. Just like primal rendering, many differentiable global illumination methods are based on some form of path tracing.

In graphics. In the graphics community, interest in differentiable rendering dates back several decades [95, 96], although the scalability and generality of the methods have improved significantly in recent years. In early examples, only direct illumination was accounted for, in order to solve for the unknowns (*e.g.* BSDF parameters) in closed form. From the start, it was clear that under-determinism would be an important challenge in inverse rendering.

The volumetric case has been investigated thoroughly over the last decade, with applications such as appearance capture of textiles [97, 98]. Later work has focused on parametric derivatives to optimize the scattering properties of volumes [99, 100, 101,

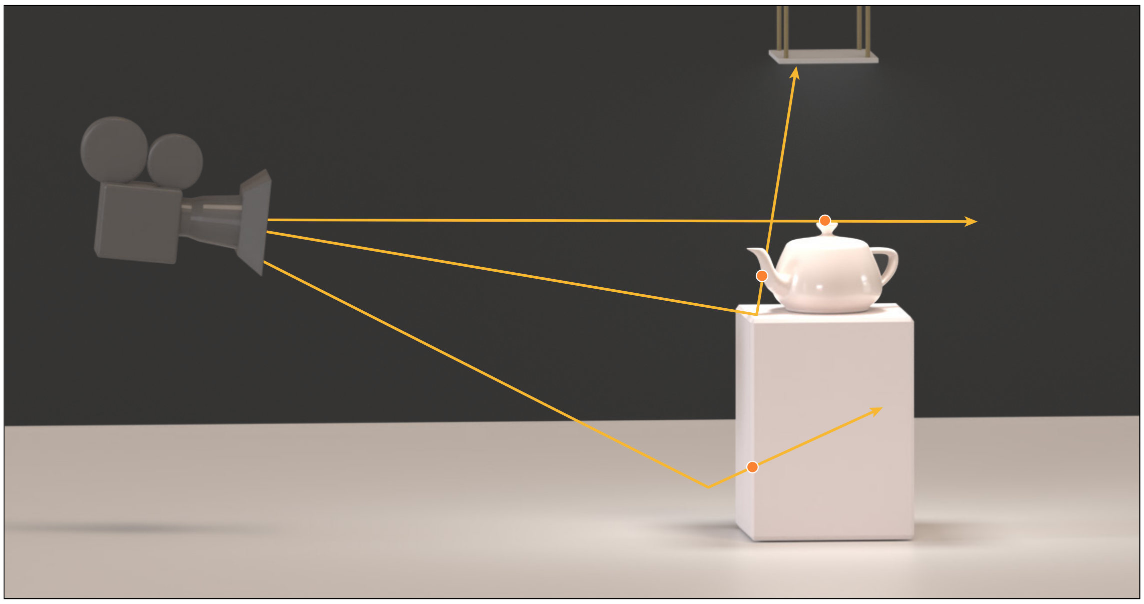


Figure 2.6: In addition to the discontinuities from instantaneous changes in primary visibility, discontinuities can appear at each step of global illumination algorithms such as path tracing (orange dots).

102], using differentiable volumetric path tracers. Zhao et al. [103] used gradients to discover *similarity relations*, *i.e.* sets of parameters that result in a similar final appearance but at a fraction of the rendering cost.

Recently, methods have been proposed to differentiate through the entire path tracing algorithm given arbitrary input scenes. Kasper et al. [104] focused on recovering distant illumination using a differentiable path tracer. Li et al. [57] presented a fully differentiable path tracer including support for discontinuities. Gradients are estimated with manually derived functions, resulting in high performance but making the system relatively hard to extend. The article's implementation was released as the *Redner* library, whose development has since continued. These recent advances coincide with the advent of hardware-accelerated ray tracing on GPUs, enabling performant implementations of path tracing on the GPU.

Inverse radiosity. Inverse radiosity (*e.g.* [105, 106]) was used to solve for near-field illumination and Lambertian materials in indoor scenes.

Parametric discontinuities. In rasterization-based methods, discontinuities are typically handled by smoothing the rasterization step (such as with analytic anti-aliasing, shown in Figure 2.5). With global illumination, the problem becomes much more difficult due to silhouette discontinuities potentially occurring at every interaction (Figure 2.6).

As noted by Zeltner et al. [12], discontinuities additionally arise within the sampling methods when using *attached* sampling techniques (see Section 3.2).

Without special handling of those discontinuities, some important gradient terms will be entirely missing. Li et al. [57] introduced the first method to correctly account for this effect in the context of physically-based rendering using a novel silhouette *edge sampling* strategy. Edge sampling works by explicitly evaluating the integrand on both sides of the discontinuity, essentially using finite differences at those locations. An acceleration data structure is built in order to efficiently sample edges based on the camera's viewpoint. One limitation of the edge sampling method is that its variance increases with the tesselation level of the mesh.

Over several articles, Zhang et al. [107, 108, 109] have built a theoretical framework for differentiable physically based rendering, including participating media. The authors apply the Reynolds transport theorem [110] to split the derivative of the light transport integral into a continuous "interior" term and a boundary term. Their formulation supports differentiating a wider class of rendering algorithms, and to build effective Monte Carlo estimators that handle complex geometric discontinuities. Discontinuities were first handled by explicit edge sampling, and later using a more efficient multi-directional sampling scheme. The key idea of this scheme is to start by sampling a boundary segment in the scene, and from there expand in both directions to form a complete path connecting the sensor to an emitter. Efficiency is improved by combining this technique with next-event estimation and bidirectional path tracing in both the prefix and suffix paths. Further improvements to the discontinuities sampling step were proposed by Yan et al. [111].

Loubet et al. [112] introduced an alternative approach: reparametrizing the light transport integral so that the new integration variable moves together with the discontinuity. The reparametrized integral no longer contains parameter-dependent discontinuities, and the appropriate gradients arise as part of the Jacobian of the change of variable. An unbiased and formalized version of this approach was then derived by Bangaru et al. [113]. Vicini et al. [114] showed that signed distance functions (SDF) admit a particularly efficient reparametrization technique.

Relation to our work. Chapters 3 and 4 discuss methods for differentiable rendering, based on automatic differentiation and the adjoint method respectively. The focus is on the efficient estimation of gradients, while still supporting global illumination. On the other hand, our algorithms do not account for parameter discontinuities, such as the one illustrated in Figure 2.6. The edge sampling and reparametrization-based methods described above are nevertheless compatible with our approach, and can be combined [12].

The recent availability of differentiable path tracing algorithms opens up new questions, such as the effectiveness of standard sampling techniques initially designed for primal rendering, when used as-is in the adjoint problem [12, 115]. Chapter 5 examines such a case for differentiable volumetric path tracing.

2.6.5 Differentiable rendering and machine learning

In recent years, a growing number of methods have combined machine learning and graphics. Some graphics techniques were augmented with learning elements, such as caching radiance for global illumination using a neural network [116]. Likewise, some computer vision methods now rely on an in-network "rendering layer", *e.g.* [117]. In such a case, the rendering component must be differentiable so that the entire pipeline can be optimized end-to-end.

Neural radiance fields (NeRFs) [118] are perhaps the most successful example of this convergence between learning and graphics. While similar methods for arbitrary viewpoint re-rendering would tend to use deeper and deeper neural networks, NeRFs impose a physically-inspired volumetric structure to the scene representation. Instead of letting the network predict an entire image directly, it needs only output the scalar density and RGB emission of a volume. Ray marching, a standard volume rendering technique, is then used to compute the volume's final appearance. Moving the rendering process out of the network and into an explicit algorithm allows the network to be much smaller. Subsequent works [119, 120] have shown that the neural network can be replaced by a simple 3D grid data structure, while preserving most of the reconstruction quality. It seems that the emissive volume representation, which is well behaved under optimization, is key to the quality of the results.

In Chapter 10, we leverage such a NeRF-inspired emissive volume to bootstrap the optimization of a physically based scattering volume, avoiding sub-optimal local minima.

2.6.6 Applications of differentiable rendering

Inverse reconstruction problems are found in a number of fields, making differentiable rendering applicable to a wide variety of image- or light transport-based applications.

Inverse rendering. First, the potential in vision-style shape and material acquisition is clear. Early examples include face reconstruction [95] and the factorization of BRDF and lighting assuming single scattering [96]. Khungurn et al. [97] and Zhao et al. [98] use micro CT scans to fit the parameters of fabric or yarn-level appearance models. Volume parameters can be estimated from images [99, 101, 121]—this is also the topic of Chapters 5 and 10. Recently, the parameters of procedural material graphs were optimized to match image examplars [122, 123, 124, 125, 126]. For single objects, joint recovery of shape and materials has been demonstrated [127]. At a larger scale, Azinović et al. [128] recovered the lighting and materials of entire rooms while accounting for global illumination. In Chapter 11, we perform such a reconstruction on real data.

These inverse rendering methods may soon become an integral part of digital content creation workflows. Usage of physically based inverse rendering methods allows for the captured materials and shapes to be repurposed, edited and relit seamlessly—as opposed to simpler methods that may merge all lighting information into the result.

Scientific applications. Beyond standard computer vision problem settings, differentiable graphics opens the door to a wide range of scientific applications. Sun et al. [129] proposed an end-to-end differentiable pipeline for lens design, optimizing the lens parameters and a neural network jointly for desirable properties such as extended depth-of-field. Nindel et al. [130] optimized the color and opacity of printed materials for more accurate color reproduction in 3D printing. In Chapter 9, we optimize the surface of a slab of glass, or a lens with a continuously-varying index of refraction, to cast light from a uniform source into arbitrary caustic patterns.

Differentiable physically based rendering has also been used to obtain sharper X-ray reconstructions for medical imaging [131], to estimate the shape and properties of clouds [132, 133, 134], and to identify the nature and pose of space debris [135].

Even though some of these problems may currently be solved using heuristic or learning-based methods, developing a physically based solution can lead to improved results and interpretability.

Primal rendering. Finally, derivatives have been put to use in the context of forward rendering, *e.g.* to estimate spatial and directional gradients for adaptive sampling and interpolation [136] or local curvature to construct better Monte Carlo Markov Chain proposal distributions [82].

Part I

Algorithms

We now turn to the physically based differentiable rendering algorithms developed over the course of this thesis.

Problem setting. Our main concern is the efficient estimation of image gradients with respect to scene parameters. Consider for example the classic inverse rendering problem illustrated in Figure 2.7: given one or more photographs of a scene which we wish to reconstruct digitally, we must recover scene parameters (object shapes and positions, lighting, material models, etc) that faithfully match the observations. Since the rendering function is much too complex to invert analytically—at least in the case of physically based rendering—we turn to *gradient-based optimization*. We start from an initial guess of the scene state, and progressively refine it according to an *objective function* which quantifies, as well as feasible, the distance between the current and goal states. Every time the scene parameters are updated after a gradient step, we render the new state of the scene. This is an *analysis by synthesis* approach.

Typical objective functions include the pixelwise L_1 and L_2 distances between the re-rendered images and the references, which are also called *photometric* losses.

Beyond standard object or scene reconstruction, many applications can benefit from efficient scene parameter gradient estimation, see Section 2.6.6.

Challenges. The *number* of parameters to optimize vary drastically depending on the application and particular scene, from *e.g.* a single roughness value to tens of millions of texture entries. For this reason, we focus on methods that support so-called *reverse-mode* or *backpropagation*. In other words, we must be able to transform gradients w.r.t. a single or few outputs (*e.g.* the objective value) to gradients w.r.t. many inputs (the scene parameters).

The main difficulty we will encounter is that the presence of global illumination and realistic material models in the scene create complex inter-dependencies between scene parameters: one parameter on an object may strongly influence the appearance of the entire scene, even if that object is not observed directly. Furthermore, we are faced with highly heterogeneous and dynamic computations. For example, a scene can contain dozens of different material models, which only become known at runtime. This is in contrast with gradient-based optimizations in the field of machine learning, where the computation is regular and mostly static.

Scope. The algorithms proposed in this thesis are restricted to the evaluation of gradients of *continuous* integrands, such as the spatially-varying albedo of a diffuse surface,

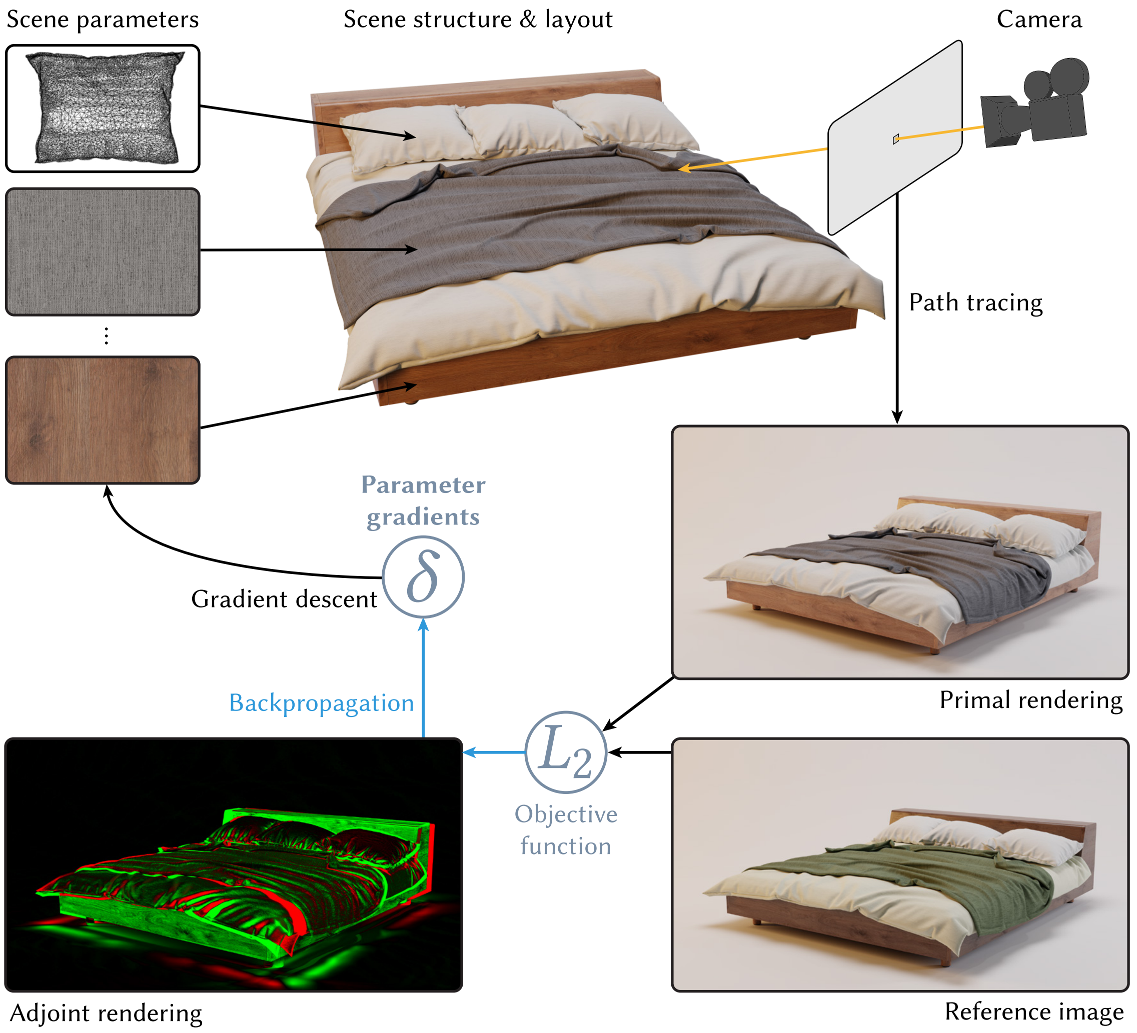


Figure 2.7: Without loss of generality, we use the following inverse rendering setting as a reference problem. We are given one or multiple reference images, which represent the goal state. We pick a starting guess for the scene state: what objects are placed in the scene, their shapes and material properties, the intensity of light sources, camera positions, etc. In the following, we propose algorithms that efficiently estimate the gradients of an arbitrary objective function w.r.t. scene parameters. This operation is labeled "backpropagation" above. We can then run a gradient descent-based stochastic optimization loop, refining the parameters until the state of the scene converges to the desired state—or at least to the local minimum closest to the initialization.

or the density of a 3D scattering volume. Other scene parameters, such as the position of mesh vertices or the camera viewpoint, induce discontinuous changes in the rendering integral. Gradients accounting for those changes will be missing from our methods' output.

These cases must be handled with methods that have been developed in concurrent work, such as edge sampling [57] and reparametrizations [112, 113, 114], discussed in Section 2.6.4. Luckily, they are orthogonal to and compatible with the algorithms presented here, and can therefore be combined [12].

Terminology. Consider a function $f(\boldsymbol{\theta})$. We refer to the estimation of $f(\boldsymbol{\theta})$ itself as the *primal* problem, and that of $\partial_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ as the *adjoint* problem. In the context of rendering, the primal is the main or "forward" rendering process which outputs an image of the scene. The adjoint is the inverse or "backward" rendering process, which estimates gradients of (typically) an objective function w.r.t. the scene parameters $\boldsymbol{\theta}$.

Furthermore, in the context of automatic differentiation, we use the term *detached* to refer to quantities that do not participate in gradient computation: they will be treated as constants. Unless specified, quantities are *attached* to the computation graph by default. Note that whether a given term is attached or detached has no effect on the result of the *primal* computation, but does impact correctness of the *adjoint* (see Figure 3.3).

Overview. We contribute two algorithms for efficient computation of gradients with respect to many scene parameters. We start by describing usage of reverse-mode automatic differentiation and its limitations in Chapter 3. We then present *radiative backpropagation* in Chapter 4, an adjoint method that recasts gradient estimation as a modified rendering problem. Finally, we introduce *differential ratio tracking* in Chapter 5, a specialized sampling technique for inverse volume rendering, which yields unbiased and low-variance gradients.

3 | Differentiable rendering with automatic differentiation

Relevant background: Sections 2.3 and 2.6.4.

Since our objective is to compute gradients of very complex functions, *automatic differentiation* is an enticing solution. Indeed, being able to simply write the rendering code (*e.g.* a path tracer) and automatically obtaining correct gradients is an ideal situation for researchers and practitioners. In fact, the availability of high-level AD-based frameworks such as TensorFlow [61] and PyTorch [62] certainly contributed to the rapid advancement of machine learning research. However, we will show that several challenges make the applicability of AD to differentiable physically based rendering limited to simple applications and that correctness is not guaranteed.

3.1 Algorithm

As we have seen in the introduction, scenes may contain tens of millions of parameters that must be optimized simultaneously. For this reason, we focus on *reverse-mode* automatic differentiation.

Chapter 7 describes the design of Mitsuba 2, a *differentiable* physically based rendering system supporting AD. A great advantage of such a system is that users can easily implement new components such as rendering algorithms, BSDFs, emitters, sensors, etc, without providing additional code dedicated to the estimation of gradients.

Interestingly, since we only require *local* derivatives and do not handle discontinuities, not all parts of the system must be subject to differentiation. In particular, we do not need to differentiate the ray tracing procedure, which represents a large part of the computational burden. It is enough to use it in a *detached* manner, as if it were a lookup function that maps rays to triangle IDs. We can then compute the local quantities such as the exact intersection point within the triangle, texture coordinates and shading normals from the ID and barycentric coordinates.

Given such a system, we can already obtain many useful gradients such as BSDF texture parameters or emitter intensity. However, extensive usage of Mitsuba 2's AD mode for inverse rendering revealed several pitfalls that we examine below.

3.2 Correctness

Although it is tempting to directly apply AD to the entire implementation of our algorithms of choice, this may lead to incorrect results.

Correlation between primal and adjoint estimators. First off, Gkioulekas et al. [99] (and later Azinović et al. [128]) have shown that in the context of Monte Carlo-based stochastic optimization, we must *decorrelate* our estimates of the primal and the adjoint. Indeed, consider for example minimizing the L_2 distance of a rendered image I to a reference image I_{ref} over scene parameters $\boldsymbol{\theta}$:

$$g(I) = \frac{1}{2}||I - I_{\text{ref}}||_2^2. \tag{3.1}$$

We take the derivative w.r.t. $\boldsymbol{\theta}$ in order to use gradient-based optimization:

$$\partial_{\boldsymbol{\theta}} g(I) = (I - I_{\text{ref}})\, \partial_{\boldsymbol{\theta}} I. \tag{3.2}$$

Unfortunately, in practice we do not have access to the true values of I and $\partial_{\boldsymbol{\theta}} I$—only noisy estimates $\hat{I}$ and $\partial_{\boldsymbol{\theta}} \hat{I}$. If there is any correlation between the two estimates, their product will *not* in general be an unbiased estimate of the true product:

$$\mathbb{E}[(\hat{I} - I_{\text{ref}})\, \partial_{\boldsymbol{\theta}} \hat{I}] \neq \mathbb{E}[\hat{I} - I_{\text{ref}}]\, \mathbb{E}[\partial_{\boldsymbol{\theta}} \hat{I}]. \tag{3.3}$$

We must then ensure that the two estimates are decorrelated. In the context of differentiable path tracing, this will involve tracing a separate set of paths in the primal and adjoint passes, rather than backpropagating directly over the primal rays. The resulting bias is illustrated in Figure 3.1 by computing gradients w.r.t. albedos of the Cornell box surfaces.

Importantly, this consideration applies to all gradient estimation methods and is not limited to AD—although the mistake is particularly easy to make with AD as the product of the correlated terms is carried out automatically by the system.

Objective with non-linear gradients. Now consider another objective function $g(I)$ such that $\partial_I\, g(I)$ is not linear, for example the L_1 loss:

$$g(I) = |I - I_{\text{ref}}|\,. \tag{3.4}$$

This non-linearity leads to another source of bias in the gradients:

$$\mathbb{E}[\frac{\partial g}{\partial I}(\hat{I})] \neq \frac{\partial g}{\partial I}(\mathbb{E}[\hat{I}]) \tag{3.5}$$

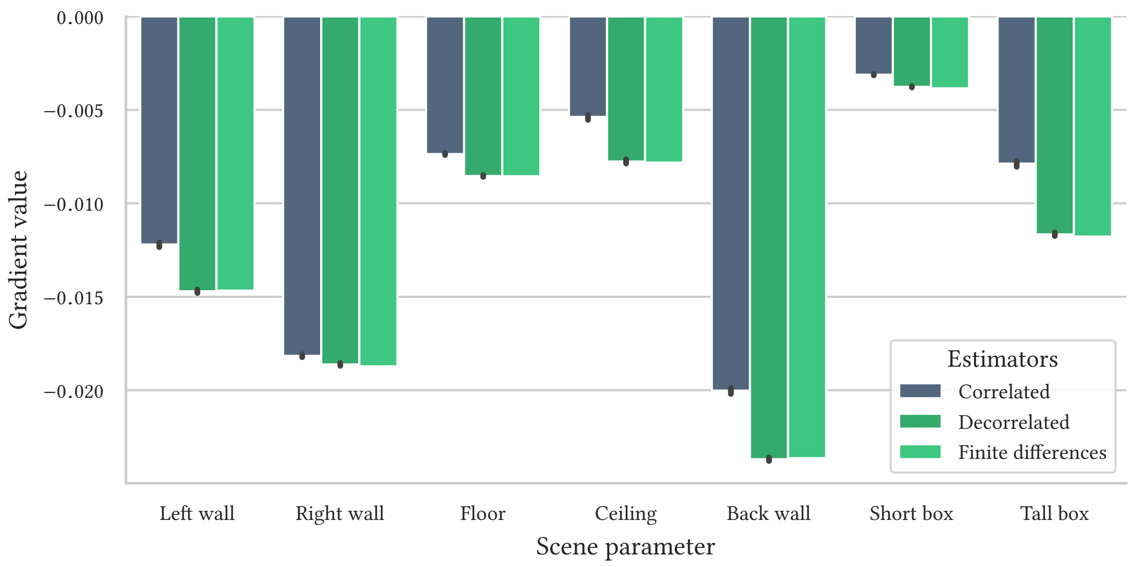


Figure 3.1: Computing gradients w.r.t. the albedo of the Cornell box's surfaces (red channel only). To test unbiasedness, gradients are estimated 64 times with 1 sample per pixel (spp) and averaged together. We first use correlated forward and adjoint estimators, which leads to bias (Equation (3.3)). By decorrelating the estimators (*i.e.* using different RNG seeds), we obtain correct gradients, matching the finite differences reference ($\epsilon = 5 \cdot 10^{-3}$).

We do not address this issue in this thesis: when an objective function with non-linear derivatives is used in our experiments, we assume that the sample count is sufficient to make the bias negligible. Nevertheless, this source of bias and its practical effects should be further investigated in future work.

Discontinuities. Most AD systems are only able to detect and handle *continuous* operations on floating point variables. In particular, control flow operations such as `if` and `while`, as well as discontinuous (delta) changes will not contribute to the gradients. This bias is illustrated in Figure 3.2 using Mitsuba 3's AD mode to estimate gradients w.r.t. the horizontal translation of an object. Shading gradients are correctly accounted for, but the effect of visibility changes at the silhouette is missing.

Discontinuities could be handled with AD by adding explicit support for delta function and integration as first-class citizens [77, 78]. However, this remains out of scope for this thesis.

Attached and detached estimators. Finally, Zeltner et al. [12] highlighted another source of bias when combining Monte Carlo estimators with AD. Consider for example

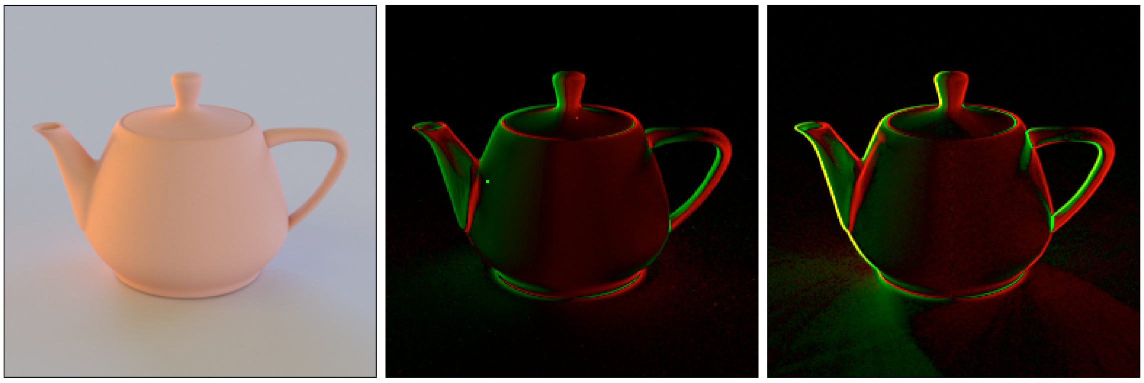

(a) Primal rendering
(b) Automatic differentiation
(c) Finite differences ($\epsilon = 5 \cdot 10^{-2}$)

Figure 3.2: Gradients from discontinuous (delta) changes are missing when using AD (bias). We visualize the effect of the horizontal translation of the teapot mesh (**a**) on the pixel values ("forward gradients"). Automatic differentiation (**b**) correctly accounts for changes to shading, but misses the silhouette (visibility) gradients. Finite differences (**c**) include all terms, including secondary effects such as changes to the teapot's shadows due to the translation.

the following infinite sum:

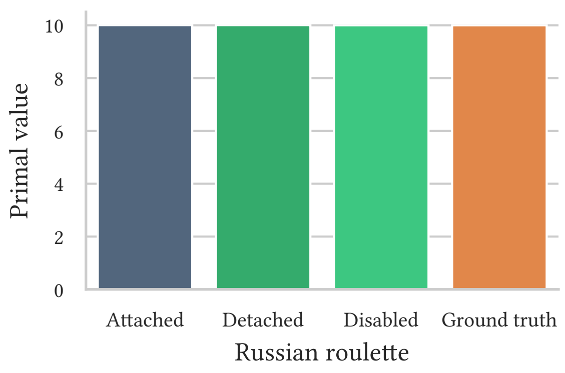
$$f(\boldsymbol{\theta}) = \sum_{i=0}^{\infty} \left(\prod_{j=1}^{i} \exp(-(t_j - t_{j-1})\, \boldsymbol{\theta}) \right) c_i, \tag{3.6}$$

It is modeled after the transmittance estimation problem in heterogeneous volumes (Section 2.2.3), where $\boldsymbol{\theta}$ plays a role analogous to the medium density σ_t. In Figure 3.3, we show that a direct application of AD to the Russian roulette (RR) estimator of this sum incorrect gradient unless special care is taken to *detach* the RR weight w. In other words, even if w involves the differentiable parameter $\boldsymbol{\theta}$, it should be treated as a constant from the perspective of the AD system. More generally, any sampling weight stemming from a discrete decision, which is not visible to the AD system, should be detached [12]. This example further confirms that we cannot blindly apply AD to complex Monte Carlo simulations.

3.3 Memory usage

The cost of maintaining a complex AD graph for highly dynamic code at runtime can be amortized by simulating many rays in parallel, effectively "widening" the graph. This is called vector-mode AD [58]. Section 7.2.3 discusses these considerations in more detail.

However, as the complexity of the computation graph—and therefore the edge count—quickly grows with the length of simulated light paths, the AD graph often saturates



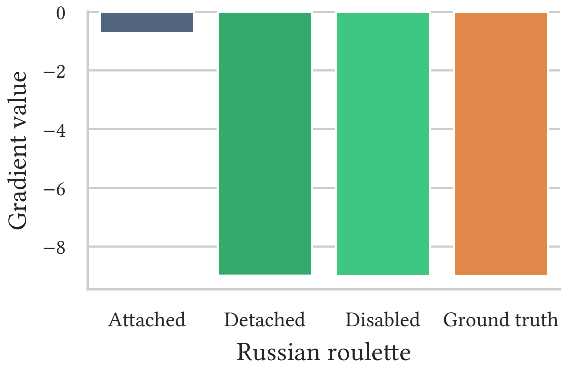


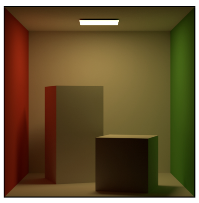
Figure 3.3: We estimate values of $f(\boldsymbol{\theta})$ from Equation (3.6) and its gradients $\partial_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ with either a fixed and large number of terms (4096), stochastically with Russian roulette, or analytically. All variants match the ground truth for the primal value $f(\boldsymbol{\theta})$. However, gradients computed with AD are incorrect when the Russian roulette continuation probability is *attached*, *i.e.* part of the computation graph.

available memory. This is especially true when running on the GPU, where memory is at a premium. The memory consumption of inverse rendering in Mitsuba 3 for two simple relatively simple test scenes is given in Table 3.1. For the trivial Cornell box scene, an excerpt of the computation graph is shown in Figure 3.4. Note these measurements already include numerous graph simplifications and optimizations.

Although it remains possible to reduce memory usage by splitting the computation into smaller batches and averaging together the resulting images or gradients, the total runtime grows linearly with the number of passes and quickly becomes impractical for most non-trivial scenes.

Table 3.1: GPU memory usage for simple AD-based inverse rendering scenarios in Mitsuba 3. The size of the AD graph becomes the main bottleneck and prevents scaling to larger problem instances.

Scene	Resolution	# params	Memory
Cornell box	256×256 @ 4spp	24	1.219 GiB
Cornell box	720×720 @ 8spp	24	19.52 GiB
Staircase	256×256 @ 4spp	55364559	2.296 GiB
Staircase	720×720 @ 8spp	55364559	20.53 GiB



3.4 Selective usage of AD

Despite the shortcomings described above, we believe there is strong value in selective applications of AD. In fact, we introduce an adjoint method in Chapter 4 which allows replacing most of the AD graph with a modified rendering problem. AD remains useful for the generation of adjoint programs computing all quantities that are not directly related to the light transport itself, such as BSDF and emitter evaluations. These adjoint programs are discussed in more detail in Chapter 8.

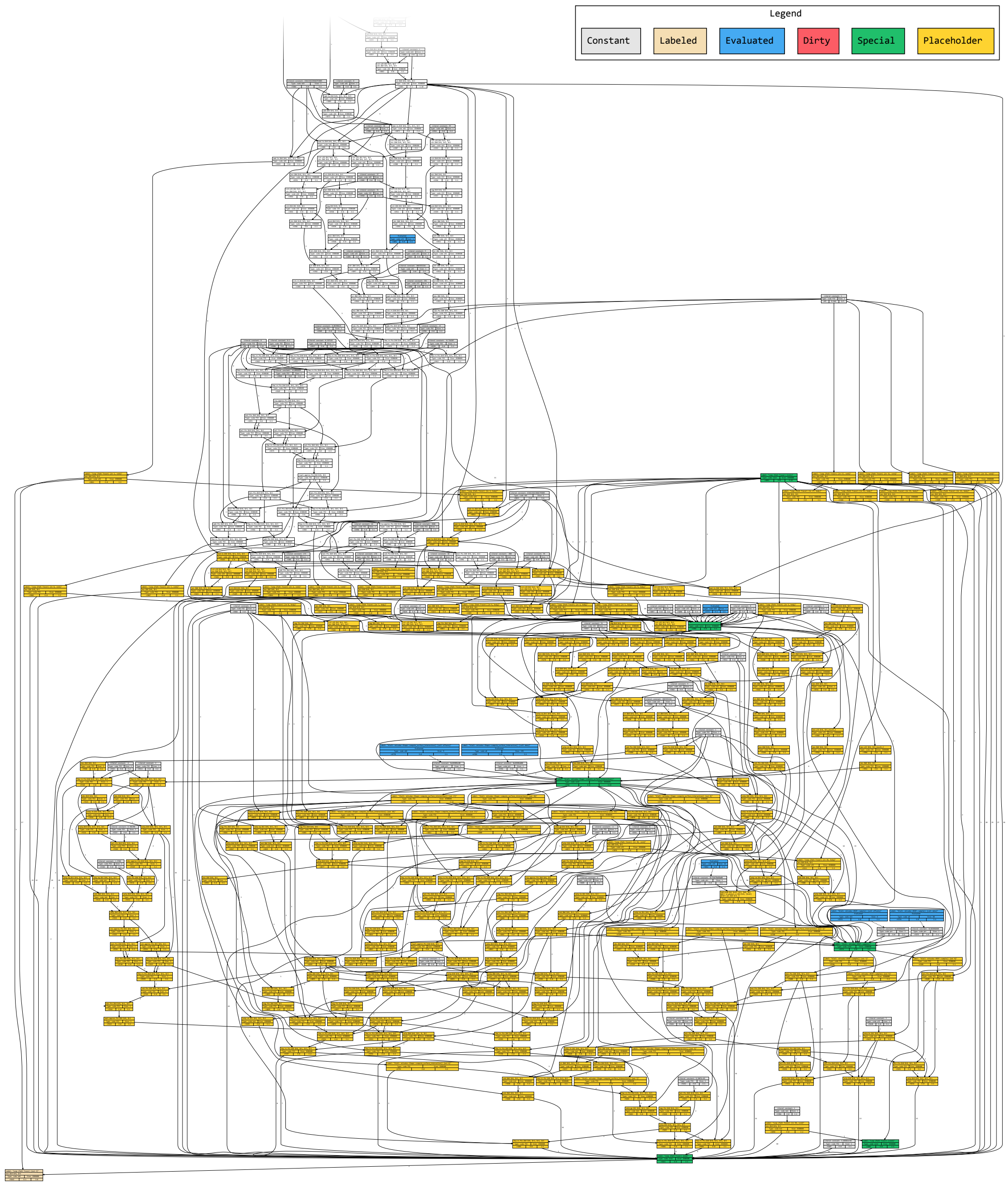


Figure 3.4: A small excerpt of the computation graph representing the (differentiable) path tracing algorithm as implemented in Mitsuba 3, when rendering the simple Cornell Box scene. Each edge stores arrays of weights with size proportional to the number of rays traced. Note that automatic graph simplifications were already applied.

4 | Radiative Backpropagation

Relevant background: Sections 2.2, 2.4 and 2.6.

4.1 Introduction

Chapter 3 established that automatic differentiation for physically based rendering suffers from fundamental scaling issues and correctness pitfalls. In this chapter, we introduce *radiative backpropagation*, an adjoint method for differentiable rendering that addresses these limitations. Despite operating in reverse-mode, our method does not require a transcript of intermediate steps, allowing it to scale to complex and long-running simulations. Our key observation is that backpropagating derivatives through a rendering algorithm can be re-cast as the solution of a modified light transport problem involving the partial derivative of radiance with respect to the optimization objective. Importantly, this new problem can be solved separately from the original rendering problem, *i.e.* without retaining any intermediate state. It also naturally permits usage of separate specialized sampling strategies for the *primal-* (forward rendering of the scene) and *adjoint* problems (estimating gradients w.r.t. scene parameters).

Our algorithm admits a vastly more efficient implementation, where automatic differentiation is applied selectively to produce “derivative shaders”. Chapter 8 details these systems-related aspects and demonstrates speedups of up to 1000× over the AD-based method of Chapter 3.

4.2 Radiative transfer

Before delving into the specifics of our method, we briefly recall the relevant radiative transfer equations of Section 2.2.2 and their differential formulations. The former roughly follows the notation of Veach [25], and the latter is a subset of the framework proposed by Zhang et al. [107], which we include here for completeness. To keep the discussion simple, we initially focus on a simplified problem without volumetric effects, and we furthermore ignore derivatives that arise due to parameter-dependent silhouette boundaries. The ideas underlying radiative backpropagation are orthogonal to these aspects, and we refer the reader to Zhang et al. [107] for a full definition of the differential quantities with all terms. Radiative backpropagation is generalized to the volumetric setting in Section 4.3.5.

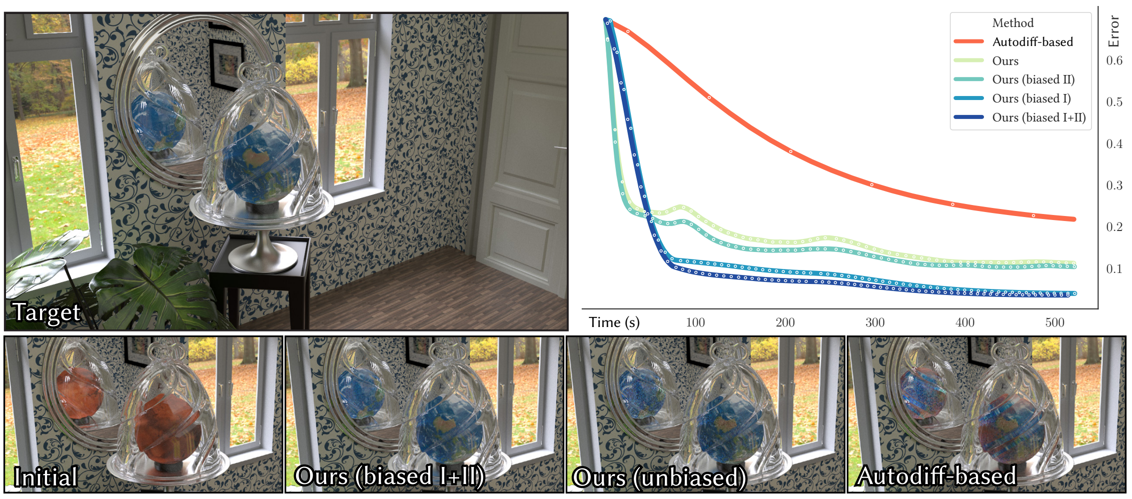


Figure 4.1: Our method is able to reconstruct the texture of a globe seen through a bell jar in this interior scene with complex materials and interreflection. Starting from a different initialization (Mars), it attempts to match a reference rendering by differentiating the L_2 image distance w.r.t. scene parameters with respect. The plot on the right shows convergence over time for prior work [14] and multiple variants of radiative backpropagation. Our method removes the severe overheads of differentiation compared to ordinary rendering, and we demonstrate speedups of up to $\sim 1000\times$ compared to prior work.

Rendering algorithms estimate sets of measurements $I_1, \dots, I_n$ corresponding to pixels in an image. These measurements are defined as inner products on ray space $\mathcal{A} \times S^2$ involving the incident radiance L_i and the importance function W_k of pixels $k = 1, \dots, n$:

$$\begin{aligned} I_k &= \langle W_k, L_i \rangle \qquad \textbf{(Measurement)} \\ &= \int_{\mathcal{A}} \int_{S^2} W_k(\mathbf{x}, \boldsymbol{\omega})\, L_i(\mathbf{x}, \boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\omega}^\perp\, \mathrm{d}\mathbf{x}. \end{aligned}$$

where $\mathrm{d}\boldsymbol{\omega}^\perp$ indicates integration with respect to projected solid angle at $\mathbf{x}$. Radiance is invariant along unoccluded rays, relating L_i to the outgoing radiance at the nearest surface $\mathbf{r}(\mathbf{x}, \boldsymbol{\omega})$ visible along the ray $(\mathbf{x}, \boldsymbol{\omega})$:

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = L_o(\mathbf{r}(\mathbf{x}, \boldsymbol{\omega}), -\boldsymbol{\omega}). \qquad \textbf{(Transport)}$$

At surfaces, radiance satisfies an energy balance condition relating sources and sinks, specifically emission $L_e(\mathbf{x}, \boldsymbol{\omega})$, and absorption or reflection of incident illumination modeled via $f_s(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}')$, the BSDF of the surface at $\mathbf{x}$:

$$L_o(\mathbf{x}, \boldsymbol{\omega}) = L_e(\mathbf{x}, \boldsymbol{\omega}) + \int_{S^2} L_i(\mathbf{x}, \boldsymbol{\omega}')\, f_s(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}')\, \mathrm{d}\boldsymbol{\omega}'^\perp. \qquad \textbf{(Scattering)}$$

Monte Carlo rendering techniques such as path tracing (Section 2.2.4) recursively sample the above equations using sophisticated importance sampling strategies to obtain unbiased estimates of the measurements $I_1, \dots, I_n$.

4.3 Method

4.3.1 Differential radiative transfer

We now turn to *differential radiative transfer* by differentiating the left and right hand sides of the preceding three equations with respect to all scene parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$, resulting in vectorial equations. For notational convenience, we define $\partial_{\boldsymbol{\theta}} := {}^{\partial}/_{\partial\boldsymbol{\theta}}$ and assume component-wise multiplication of vector quantities. The resulting equations will relate *differential incident, outgoing, and emitted radiance* $\partial_{\boldsymbol{\theta}} L_i$, $\partial_{\boldsymbol{\theta}} L_o$, and $\partial_{\boldsymbol{\theta}} L_e$. Differential radiance in many ways resembles "ordinary" radiance, and we will simply think of it as another type of radiation that can be transported and scattered.

To start, a differential measurement of pixel k involves a ray-space inner product involving differential incident radiance $\partial_{\boldsymbol{\theta}} L_i$ and importance W_k:

$$\partial_{\boldsymbol{\theta}} I_k = \int_{\mathcal{A}} \int_{S^2} W_k(\mathbf{x}, \boldsymbol{\omega}) \partial_{\boldsymbol{\theta}} L_i(\mathbf{x}, \boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{\omega}^{\perp} \, \mathrm{d}\mathbf{x}. \qquad \textbf{(Differential measurement)}$$

Here, we have assumed a static sensor[1], hence $\partial_{\boldsymbol{\theta}} W_k = 0$. Differentiating the transport equation yields no surprises: differential radiance is transported in the same manner as normal radiance:

$$\partial_{\boldsymbol{\theta}} L_i(\mathbf{x}, \boldsymbol{\omega}) = \partial_{\boldsymbol{\theta}} L_o(\mathbf{r}(\mathbf{x}, \boldsymbol{\omega}), -\boldsymbol{\omega}). \qquad \textbf{(Differential transport)}$$

The derivative of the scattering equation is more interesting:

$$\begin{aligned}\partial_{\boldsymbol{\theta}} L_o(\mathbf{x}, \boldsymbol{\omega}) = {} & \underbrace{\partial_{\boldsymbol{\theta}} L_e(\mathbf{x}, \boldsymbol{\omega})}_{\textbf{Term 1}} \qquad \textbf{(Differential scattering)} \\ & + \int_{S^2} \Big[\underbrace{\partial_{\boldsymbol{\theta}} L_i(\mathbf{x}, \boldsymbol{\omega}') \, f_s(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}')}_{\textbf{Term 2}} + \underbrace{L_i(\mathbf{x}, \boldsymbol{\omega}') \, \partial_{\boldsymbol{\theta}} f_s(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}')}_{\textbf{Term 3}} \Big] \, \mathrm{d}\boldsymbol{\omega}'^{\perp}.\end{aligned}$$

The above can be interpreted as another kind of energy balance equation. In particular,

- **Term 1.** Differential radiance is "emitted" by light sources whose brightness changes when perturbing the scene parameters $\boldsymbol{\theta}$.
- **Term 2.** Differential radiance "scatters" in the same way as normal radiance, *i.e.* according to the BSDF of the underlying surface.

[1]Compatibility of our approach with existing methods which compute visibility-related gradients is discussed in Section 4.5.

- **Term 3.** Differential radiance is additionally "emitted" when the material at $\mathbf{x}$ changes as a function of the scene parameters.

In the following, we will develop tools to sample these equations in reverse mode.

4.3.2 Optimization using differential transport

Applications of differentiable rendering to optimization problems (*e.g.* inverse rendering) require an objective function $g : \mathcal{Y} \to \mathbb{R}$ that measures the quality of tentative solutions. This objective could be a simple pixel-wise L_1 or L_2 error or a function with a more complex dependence, such as a Wasserstein metric or a convolutional neural network. Given g, we seek to minimize the composition $g(f(\boldsymbol{\theta}))$ using an iterative gradient-based optimization technique, bearing in mind that evaluations are necessarily noisy due to the underlying Monte Carlo integration.

It is interesting to note that gradients are a major asset in this context: Jamieson et al. [137] showed that *derivative free optimization* (DFO) methods can never achieve an optimization error better than $\Omega(1/\sqrt{l})$ when optimizing noisy objectives, where l is the iteration count, and this even applies to methods that estimate derivatives using finite differences. In contrast, the error of gradient-based methods always decreases proportionally to $\Theta(1/l)$ when the function is strongly convex[2].

The first issue with this approach was noted by Gkioulekas et al. [99] and can be observed after rewriting the gradient as a product of the Jacobians via the chain rule: Since this occurs simultaneously using the same samples, the resulting random variables are correlated, meaning that the identity $\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]$ no longer holds, and the resulting gradients are thus biased. The second problem is that the transcript of $g \circ f$ is *even longer* than the already problematic rendering step, especially when the objective function is nontrivial. In contrast to AD-based gradient computation, our approach splits differentiation into three steps: rendering, differentiation of the objective, and radiative backpropagation. In pseudocode:

[2]A standard assumption made to analyze the asymptotic behavior of such methods.

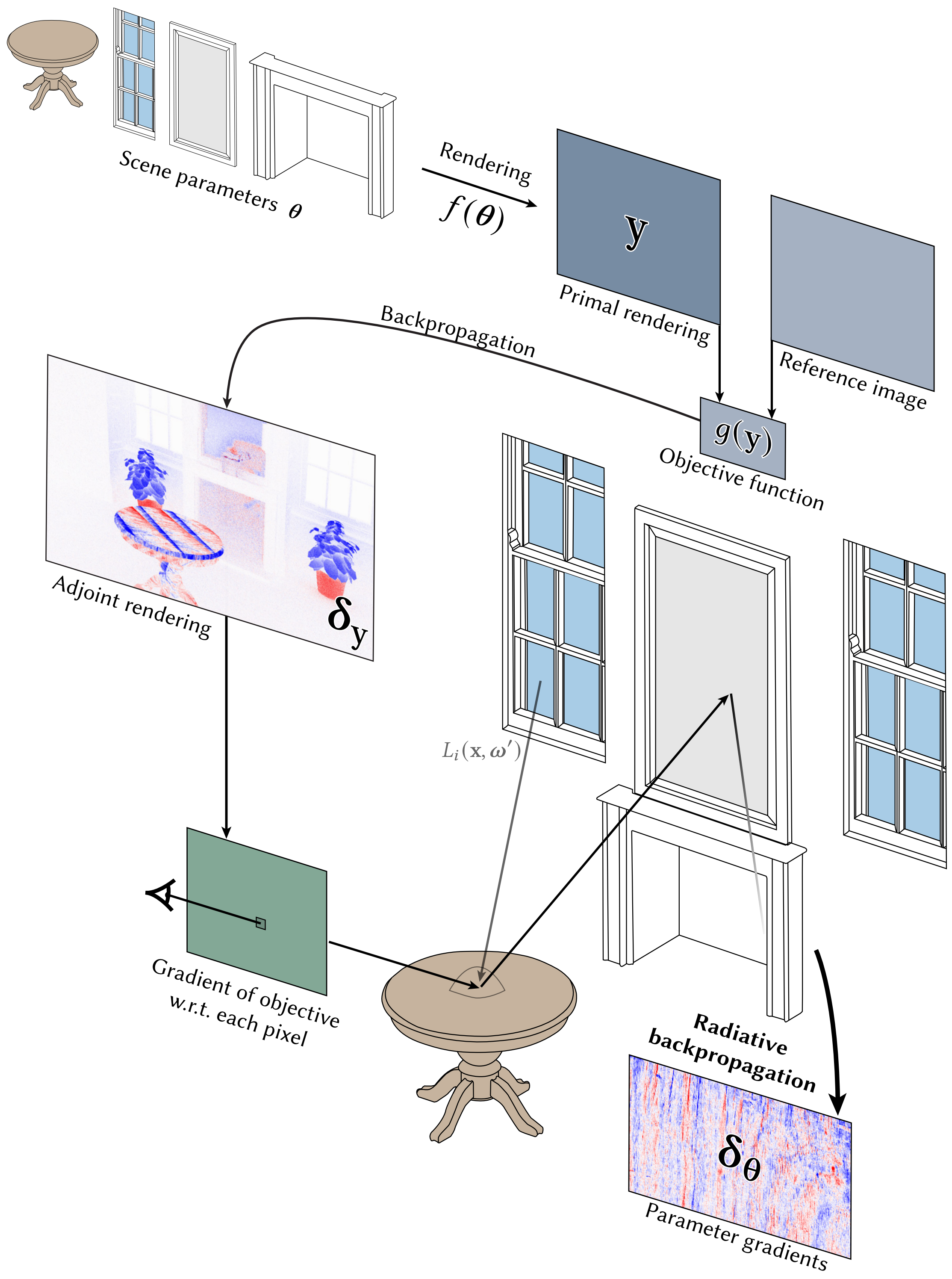


Figure 4.2: Our method efficiently computes gradients of an arbitrary objective function $g(f(\boldsymbol{\theta}))$ w.r.t. scene parameters $\boldsymbol{\theta}$. At each iteration, it performs a fast primal (*i.e.* non-differentiable) rendering step producing an image $\mathbf{y} = f(\boldsymbol{\theta})$. The second step differentiates the objective function to compute an "adjoint rendering" $\boldsymbol{\delta}_\mathbf{y}$ that expresses the sensitivity of individual image pixels w.r.t. the optimization task. Radiative backpropagation is the final step and the main contribution of this chapter. It consists of a physical simulation, in which adjoint radiance $\boldsymbol{\delta}_\mathbf{y}$ is emitted by the sensor, scattered by the scene, and eventually received by objects with differentiable parameters. Its output are parameter gradients $\boldsymbol{\delta}_{\boldsymbol{\theta}}$.

```
def grad($\boldsymbol{\theta}$):
    # 1. Ordinary rendering (no AD)
    y = f($\boldsymbol{\theta}$)
    # 2. Differentiate objective at y (manually or w/ AD)
    $\boldsymbol{\delta}_\mathrm{y}$ = $\mathbf{J}_g^T(\mathrm{y})$
    # Estimate $\boldsymbol{\delta}_\theta = \mathbf{J}_f^T \boldsymbol{\delta}_\mathrm{y}$ using radiative backpropagation
    return radiative_backprop($\boldsymbol{\theta}$, $\boldsymbol{\delta}$y)
```

We refer to $\boldsymbol{\delta}_\mathrm{y} \in \mathbb{R}^n$ as the *adjoint rendering*[3]. It encodes the sensitivity of pixels with respect to the objective, *i.e.* how the rendered image should change to optimally improve the objective locally. The algorithm's main steps are illustrated in Figure 4.2.

4.3.3 Adjoint radiance

We now turn to the stochastic evaluation of $\mathbf{J}_f^T \boldsymbol{\delta}_\mathrm{y}$. Recall that the rows of the Jacobian $\mathbf{J}_f$ are the parametric derivatives of pixel measurements, *i.e.*

$$\mathbf{J}_f^T = \left[\partial_\theta I_0, \ldots, \partial_\theta I_n\right].$$

Substituting the differential measurement equation yields

$$\begin{aligned}\mathbf{J}_f^T \boldsymbol{\delta}_\mathrm{y} &= \sum_{k=1}^{n} \boldsymbol{\delta}_{\mathrm{y},k}\, \partial_\theta I_k \\ &= \int_{\mathcal{A}} \int_{S^2} \underbrace{\left[\sum_{k=1}^{n} \boldsymbol{\delta}_{\mathrm{y},k} W_k(\mathbf{x}, \boldsymbol{\omega})\right]}_{=: A_e(\mathbf{x},\boldsymbol{\omega})} \partial_\theta L_i(\mathbf{x}, \boldsymbol{\omega})\, \mathrm{d}\boldsymbol{\omega}^\perp\, \mathrm{d}\mathbf{x},\end{aligned}$$

where we have defined the *emitted adjoint radiance* $A_e(\mathbf{x}, \boldsymbol{\omega})$ and $\boldsymbol{\delta}_{\mathrm{y},k}$ refers to pixel k of $\boldsymbol{\delta}_\mathrm{y}$. With this substitution, the desired gradient $\mathbf{J}_f^T\, \boldsymbol{\delta}_\mathrm{y}$ turns into an inner product on ray space:

$$\mathbf{J}_f^T \boldsymbol{\delta}_\mathrm{y} = \langle A_e, \partial_\theta L_i \rangle.$$

We think of A_e as an emitted quantity—for instance, in the case of a pinhole camera, it can be interpreted as a textured "spot light" that projects the adjoint rendering into the scene. Whereas a primal renderer might, *e.g.* estimate an inner product of emitted importance and incident radiance, radiative backpropagation replaces these with differential quantities: emitted adjoint radiance, and the incident differential radiance.

[3]Unless noted otherwise, we use the qualifier "adjoint" in the AD sense throughout this chapter, *i.e.* indicating sensitivity with regards to an optimization objective. The terminological overlap with bidirectional light transport techniques is unfortunate.

4.3.4 Operator formulation

The previous expression is reminiscent of the starting point of the operator formulation developed by Arvo [138] and Veach [25]. We wish to follow a similar approach here and begin by defining an effective emission term $\mathbf{Q}(\mathbf{x}, \boldsymbol{\omega})$ that includes terms 1 and 3 of the differential scattering equation from Section 4.3.1. Recall that the latter is nonzero when the scattered radiance at $\mathbf{x}$ changes as a function of the material parameters.

$$\mathbf{Q}(\mathbf{x}, \boldsymbol{\omega}) := \partial_\theta L_e(\mathbf{x}, \boldsymbol{\omega}) + \int_{S^2} L_i(\mathbf{x}, \boldsymbol{\omega}')\, \partial_\theta f_s(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}')\, \mathrm{d}\boldsymbol{\omega}'^{\perp}, \tag{4.1}$$

We then define a *scattering operator* $\mathcal{K}$, and a *propagation operator* $\mathcal{G}$:

$$\begin{aligned}(\mathcal{K}h)(\mathbf{x}, \boldsymbol{\omega}) &:= \int_{S^2} h(\mathbf{x}, \boldsymbol{\omega}')\, f_s(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\omega}')\; \mathrm{d}\boldsymbol{\omega}'^{\perp},\\ (\mathcal{G}h)(\mathbf{x}, \boldsymbol{\omega}) &:= h(\mathbf{r}(\mathbf{x}, \boldsymbol{\omega}), -\boldsymbol{\omega}),\end{aligned}$$

reducing the differential transport and scattering equations to

$$\partial_\theta L_i = \mathcal{G}\partial_\theta L_o, \quad \text{and} \quad \partial_\theta L_o = \mathbf{Q} + \mathcal{K}\partial_\theta L_i.$$

Differential radiance scatters and propagates like "ordinary" radiance, hence $\mathcal{K}$ and $\mathcal{G}$ are *identical* to Veach's operators, allowing us to immediately state the solution of outgoing differential radiance:

$$\begin{aligned}\partial_\theta L_o &= \mathbf{Q} + \mathcal{K}\mathcal{G}\, \partial_\theta L_o\\ &= \underbrace{(I - \mathcal{K}\mathcal{G})^{-1}}_{=:\mathcal{S}} \mathbf{Q}\\ &= \sum_{i=0}^{\infty} (\mathcal{K}\mathcal{G})^i \mathbf{Q}\end{aligned}$$

via the solution operator $\mathcal{S}$ given in terms of a Neumann series expansion of $\mathcal{K}$ and $\mathcal{G}$. Veach also showed that $\mathcal{G}$, $\mathcal{K}$, and $\mathcal{G}\mathcal{S}$ are *self-adjoint*[4] linear operators when the scene satisfies elementary physical constraints—in particular, energy-conserving and reciprocal BSDFs. Self-adjoint operators O have the property that $\langle O v_1, v_2\rangle = \langle v_1, O v_2\rangle$, meaning that

$$\mathbf{J}^T \boldsymbol{\delta}_\mathrm{y} = \langle A_e, \partial_\theta L_i\rangle = \langle A_e, \mathcal{G}\mathcal{S}\mathbf{Q}\rangle = \boxed{\langle \mathcal{G}\mathcal{S} A_e, \mathbf{Q}\rangle}.$$

This equation encapsulates the key idea of radiative backpropagation. It states that instead of scattering and propagating differential radiance we can also start "from the other

[4]In the sense of functional analysis.

side" and scatter and propagate adjoint radiance instead. This is a *vastly easier* problem: A_e is scalar-valued, while $\mathbf{Q}$ is a vectorial function whose dimension matches that of the parameter space Θ (*i.e.* potentially millions).

4.3.5 Volumetric transport

To analyze the volumetric case, we import the operator formulation of Jakob [53]. Its generalized scattering operator is given by

$$(\bar{\mathcal{K}}h)(\mathbf{x},\boldsymbol{\omega}) := \begin{cases} \int_{S^2} h(\mathbf{x},\boldsymbol{\omega}')\, f_s(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}'^{\perp}, & \mathbf{x}\in\mathcal{A} \\ \sigma_s(\mathbf{x})\int_{S^2} h(\mathbf{x},\boldsymbol{\omega}')\, f_p(\mathbf{x},-\boldsymbol{\omega},\boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}', & \mathbf{x}\notin\mathcal{A} \end{cases}$$

which turns incident radiance into either outgoing radiance on surfaces, and into outgoing radiance per unit length in volumes. Here, σ_s is the medium's scattering coefficient and f_p is the phase function[5]. The propagation operator reads

$$(\bar{\mathcal{G}}h)(\mathbf{x},\boldsymbol{\omega}) := \int_{\mathbf{x}}^{\mathbf{r}(\mathbf{x},\boldsymbol{\omega})} \mathrm{T}(\mathbf{x},\mathbf{x}')\, h(\mathbf{x}',-\boldsymbol{\omega})\,\mathrm{d}\mathbf{x}' + \mathrm{T}(\mathbf{x},\mathbf{r}(\mathbf{x},\boldsymbol{\omega}))\, h(\mathbf{r}(\mathbf{x},\boldsymbol{\omega}),-\boldsymbol{\omega}),$$

and returns incident radiance due to both outgoing surface radiance and outgoing radiance per unit length in volumes. The function T is the volumetric transmittance defined as

$$\mathrm{T}(\mathbf{a},\mathbf{b}) = e^{-\int_{\mathbf{a}}^{\mathbf{b}} \sigma_t(\mathbf{x})\,\mathrm{d}\mathbf{x}},$$

and references the extinction coefficient σ_t. Using the above definitions, the generalized equilibrium equation[6] reads

$$L_i = \bar{\mathcal{G}}\,(\bar{\mathcal{K}}L_i + L_e).$$

[5]Note that its incident argument follows a different sign convention than the BSDF.

[6]The order of operators is reversed compared to the surface case. Details can be found in Jakob's Ph.D. thesis [53], page 57.

Differentiating the operators with respect to scene parameters via the product rule yields the following sum of terms:

$$\partial_\theta(\bar{\mathcal{K}}h)(\mathbf{x},\boldsymbol{\omega}) = \begin{cases} \displaystyle\int_{S^2} \big[\partial_\theta h(\mathbf{x},\boldsymbol{\omega}')\, f_s(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}') \\ \qquad + h(\mathbf{x},\boldsymbol{\omega}')\,\partial_\theta f_s(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}')\,\big]\,\mathrm{d}\boldsymbol{\omega}'^{\perp}, & \mathbf{x}\in\mathcal{A} \\ \displaystyle\int_{S^2} \big[\partial_\theta h(\mathbf{x},\boldsymbol{\omega}')\,\sigma_s(\mathbf{x})\, f_p(\mathbf{x},-\boldsymbol{\omega},\boldsymbol{\omega}') \\ \qquad + h(\mathbf{x},\boldsymbol{\omega}')\,\partial_\theta\sigma_s(\mathbf{x})\, f_p(\mathbf{x},-\boldsymbol{\omega},\boldsymbol{\omega}') \\ \qquad + h(\mathbf{x},\boldsymbol{\omega}')\,\sigma_s(\mathbf{x})\,\partial_\theta f_p(\mathbf{x},-\boldsymbol{\omega},\boldsymbol{\omega}')\big]\,\mathrm{d}\boldsymbol{\omega}', & \mathbf{x}\notin\mathcal{A} \end{cases}$$

and the derivative of the propagation operator is given by

$$\begin{aligned}\partial_\theta(\bar{\mathcal{G}}h)(\mathbf{x},\boldsymbol{\omega}) = &\int_{\mathbf{x}}^{\mathbf{r}(\mathbf{x},\boldsymbol{\omega})} \big[\partial_\theta \mathrm{T}(\mathbf{x},\mathbf{x}')\, h(\mathbf{x}',-\boldsymbol{\omega}) + \mathrm{T}(\mathbf{x},\mathbf{x}')\,\partial_\theta h(\mathbf{x}',-\boldsymbol{\omega})\big]\,\mathrm{d}\mathbf{x}' \\ &+ \partial_\theta \mathrm{T}(\mathbf{x},\mathbf{r}(\mathbf{x},\boldsymbol{\omega}))\, h(\mathbf{r}(\mathbf{x},\boldsymbol{\omega}),-\boldsymbol{\omega}) \\ &+ \mathrm{T}(\mathbf{x},\mathbf{r}(\mathbf{x},\boldsymbol{\omega}))\,\partial_\theta h(\mathbf{r}(\mathbf{x},\boldsymbol{\omega}),-\boldsymbol{\omega}).\end{aligned}$$

We observe that both equations contain certain terms with the factor $\partial_\theta h$, and that the removal of all other terms would yield expressions that match ordinary scattering and propagation operators applied to the function $\partial_\theta h$. These other terms only depend on primal quantities (in particular, h) and derivatives of material properties (*e.g.* $\partial_\theta\sigma_s(\mathbf{x})$), and the differential form of the operators can thus be cast into the form

$$\begin{aligned}\partial_\theta\bar{\mathcal{K}}h &= \bar{\mathcal{K}}\partial_\theta h + \bar{Q}_1 h \\ \partial_\theta\bar{\mathcal{G}}h &= \bar{\mathcal{G}}\partial_\theta h + \bar{Q}_2 h\end{aligned}$$

where the operators $\bar{Q}_1$ and $\bar{Q}_2$ model differential radiance that is emitted because optical properties depend on scene parameters $\boldsymbol{\theta}$. More specifically, $\bar{Q}_1$ describes differential emission due to perturbations in BSDFs, phase functions, and the scattering coefficient, and $\bar{Q}_2$ contains differential emission due to perturbations in transmission along ray segments:

$$(\bar{Q}_1 h)(\mathbf{x},\boldsymbol{\omega}) = \begin{cases} \displaystyle\int_{S^2} h(\mathbf{x},\boldsymbol{\omega}')\,\partial_\theta f_s(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}'^{\perp}, & \mathbf{x}\in\mathcal{A} \\ \displaystyle\int_{S^2} \big[h(\mathbf{x},\boldsymbol{\omega}')\,\partial_\theta\sigma_s(\mathbf{x})\, f_p(\mathbf{x},-\boldsymbol{\omega},\boldsymbol{\omega}') \\ \qquad + h(\mathbf{x},\boldsymbol{\omega}')\,\sigma_s(\mathbf{x})\,\partial_\theta f_p(\mathbf{x},-\boldsymbol{\omega},\boldsymbol{\omega}')\big]\,\mathrm{d}\boldsymbol{\omega}', & \mathbf{x}\notin\mathcal{A} \end{cases}$$

$$(\bar{Q}_2 h)(\mathbf{x},\boldsymbol{\omega}) = \int_{\mathbf{x}}^{\mathbf{r}(\mathbf{x},\boldsymbol{\omega})} \partial_\theta \mathrm{T}(\mathbf{x},\mathbf{x}')\, h(\mathbf{x}',-\boldsymbol{\omega})\,\mathrm{d}\mathbf{x}' + \partial_\theta \mathrm{T}(\mathbf{x},\mathbf{r}(\mathbf{x},\boldsymbol{\omega}))\, h(\mathbf{r}(\mathbf{x},\boldsymbol{\omega}),-\boldsymbol{\omega}).$$

Both terms can be sampled using standard volumetric path tracing techniques, although we will show in Chapter 5 that a dedicated sampling technique is needed to estimate unbiased and low-variance in-scattering gradients. We can now derive the differential equilibrium equation:

$$\begin{aligned}
\partial_\theta L_i &= \partial_\theta \bar{\mathcal{G}}(\bar{\mathcal{K}} L_i + L_e) \\
&= \bar{\mathcal{G}} \partial_\theta (\bar{\mathcal{K}} L_i + L_e) + \bar{Q}_2(\bar{\mathcal{K}} L_i + L_e) \\
&= \bar{\mathcal{G}}(\bar{\mathcal{K}} \partial_\theta L_i + \partial_\theta L_e + \bar{Q}_1 L_i) + \bar{Q}_2(\bar{\mathcal{K}} L_i + L_e) \\
&= \bar{\mathcal{G}}\bar{\mathcal{K}} \partial_\theta L_i + \underbrace{\bar{\mathcal{G}} \partial_\theta L_e + \bar{\mathcal{G}}\bar{Q}_1 L_i + \bar{Q}_2(\bar{\mathcal{K}} L_i + L_e)}_{=:\bar{\mathbf{Q}}} \\
&= \underbrace{(I - \bar{\mathcal{G}}\bar{\mathcal{K}})^{-1}}_{=\bar{\mathcal{S}}} \bar{\mathbf{Q}}.
\end{aligned}$$

where $\bar{\mathcal{S}}$ is the generalized solution operator and $\bar{\mathbf{Q}}$ is an effective emission term that accounts for all sources of emitted differential radiance. As before, differential radiative transfer can thus be understood as the solution to an ordinary transport problem with a modified emission term. The volumetric form of radiative backpropagation then exploits the self-adjoint nature of $\bar{\mathcal{S}}$ to efficiently compute the following inner product:

$$\mathbf{J}^T \boldsymbol{\delta}_{\mathrm{y}} = \langle A_e, \partial_\theta L_i \rangle = \langle A_e, \bar{\mathcal{S}}\bar{\mathbf{Q}} \rangle = \boxed{\langle \bar{\mathcal{S}} A_e, \bar{\mathbf{Q}} \rangle.}$$

4.3.6 Sampling strategies for differential rendering

To complete the symmetry, we can finally define analogous incident and outgoing variants of adjoint radiance satisfying

$$A_i = \mathcal{G} A_o, \quad \text{and} \quad A_o = A_e + \mathcal{K} A_i$$

In the next section, we present a simple path tracing-style integrator that samples adjoint incident radiance A_i at surface locations to compute its inner product $\langle A_i, \mathbf{Q} \rangle$ with the effective differential emission. At specific surface positions, the $\mathbf{Q}$ term is extremely sparse[7], admitting a highly efficient integration procedure.

A great variety of alternative sampling schemes are conceivable: for instance, certain scene parameters may have a significant effect on the scene's local radiance distribution (corresponding to very large component values in the effective emission term $\mathbf{Q}$). Scene

[7]On a surface with a textured diffuse BSDF, it will *e.g.* be zero except for components corresponding to texels that are interpolated during BSDF evaluations at $\mathbf{x}$.

locations affected by this could be sampled in a more targeted fashion to create an additional connection strategy akin to *next event estimation* in the context of path tracing. The benefits of such a connection strategy will be marginal if most of the generated samples cannot be connected to the adjoint subpath sampled from the camera end (*e.g.* due to occlusion). Analogous to *bidirectional path tracing* [49], it may be helpful to scatter and transport these samples multiple times through the scene to increase the probability of a successful connection, creating a family of bidirectional connection strategies.

One of the main contributions of our work is that it casts the problem of reverse-mode differentiable rendering into familiar light transport terms, enabling the application of a large toolbox of algorithms and sampling strategies developed in the last decades.

4.3.7 Radiative backpropagation path tracing

Listings 2 and 3 provide the pseudocode of a simple path tracing-style variant of radiative backpropagation. Note that all scene elements (BSDFs, emitters) in these listings are assumed to have an implicit dependence on the scene parameters $\boldsymbol{\theta}$. For simplicity, we omit a number of optimizations that are standard in path tracers, and which are similarly straightforward to implement in our method—in particular: Russian roulette, direct illumination sampling strategies for emitters, and multiple importance sampling to combine BSDF and emitter sampling strategies. Our implementation does use these optimizations.

One interesting optimization opportunity that we currently do not exploit is that many pixels may be associated with a relatively small amount of adjoint radiance (*i.e.* $A_e(\mathbf{x}, \boldsymbol{\omega}) \approx 0$), in which case rays $(\mathbf{x}, \boldsymbol{\omega})$ should be sampled directly from A_e instead of the sensor's importance function. Two lines in Listing 3 of the form:

```
grad += adjoint([[ $q(\mathbf{z})$ ]], $\boldsymbol{\delta}$)
```

deserve further explanation. This syntax indicates an evaluation of the adjoint of q: we wish to propagate the gradient $\boldsymbol{\delta}$ from the outputs of the function q evaluated at $(\boldsymbol{\theta}, \mathbf{z})$ to its differentiable parameters $\boldsymbol{\theta} \in \Theta$ by evaluating $\mathbf{J}_q^T(\boldsymbol{\theta}, \mathbf{z})\,\boldsymbol{\delta}$. The function `adjoint` then returns the derivative with respect to the scene parameters, *i.e.* a vector of size $\dim(\Theta)$. Note that adjoints of typical components of rendering systems yield *extremely sparse* gradient vectors: for instance, evaluations of a textured BSDF will only depend on a few parameters that correspond to nearby texels. Adding million-dimensional vectors with only a few nonzero entries at every interaction would be very wasteful, hence it is crucial that `adjoint()` exploits the underlying sparsity pattern. Chapter 8 discusses our implementation of this important operation.

```
def radiative_backprop($\boldsymbol{\theta}$, $\boldsymbol{\delta}_\mathrm{y}$):
    # Initialize parameter gradient(s) to zero
    $\boldsymbol{\delta}_\theta$ = 0
    for _ in range(num_samples):
        # Importance sample a ray from the sensor
        x, $\boldsymbol{\omega}$, weight = sensor.sample_ray()
        # Evaluate the adjoint emitted radiance
        weight *= $A_e(\boldsymbol{\delta}_\mathrm{y}, \mathbf{x}, \boldsymbol{\omega})$ / num_samples
        # Propagate adjoint radiance into the scene
        $\boldsymbol{\delta}_\theta$ += radiative_backprop_sample($\boldsymbol{\theta}$, x, $\boldsymbol{\omega}$, weight)
    # Finished, return gradients
    return $\boldsymbol{\delta}_\theta$
```

Listing 2: Radiative backpropagation takes scene parameters $\boldsymbol{\theta}$ and an adjoint rendering $\boldsymbol{\delta}_\mathrm{y}$ as input. It samples a large set of camera rays and propagates the associated adjoint radiance A_e (depending on $\boldsymbol{\delta}_y$) into the scene.

```
def radiative_backprop_sample($\boldsymbol{\theta}$, x, $\boldsymbol{\omega}$, weight):
    # Find an intersection with the scene geometry
    $\mathbf{x}'$ = $\mathbf{r}(\mathbf{x}, \boldsymbol{\omega})$
    # Backpropagate to parameters of emitter, if any
    $\boldsymbol{\delta}_\theta$ = adjoint([[ $L_e(\mathbf{x}', -\boldsymbol{\omega})$ ]], weight)
    # Sample a ray from the BSDF
    $\boldsymbol{\omega}'$, bsdf_value, bsdf_pdf = sample $f_s(\mathbf{x}', -\boldsymbol{\omega}, \cdot)$
    # Backpropagate to parameters of BSDF, if any
    $\boldsymbol{\delta}_\theta$ += adjoint([[ $f_s(\mathbf{x}', -\boldsymbol{\omega}, \boldsymbol{\omega}')$ ]], weight * $L_i(\mathbf{x}, \boldsymbol{\omega}')$ / bsdf_pdf)
    # Recurse
    return $\boldsymbol{\delta}_\theta$ + radiative_backprop_sample(
        $\boldsymbol{\theta}$, $\mathbf{x}'$, $\boldsymbol{\omega}'$, weight * bsdf_value / bsdf_pdf)
```

Listing 3: Simplified pseudocode of the propagation operation for surfaces. The two `adjoint()` operations correspond to the two terms of the effective differential emission Q and propagate adjoint radiance to parameters of the emitters and material models (see Section 4.3.7 for details on their semantics).

4.3.8 Worse is better? Acceleration using biased gradients.

The previous subsection introduced an unbiased algorithm for estimating parametric derivatives using an optical analog of reverse-mode propagation.

Quadratic cost. One potential stumbling block is that the effective emission term $\mathbf{Q}$ contains the primal incident radiance L_i. In the unbiased algorithm, sampling $\mathbf{Q}$ therefore involves a recursive invocation of a classical path tracer. We perform this recursive estimate at every interaction with differentiable parameters, which can be costly (quadratic complexity) when sampling long light paths with many interactions. The same issue was also reported by Zhang et al. [107] in the context of forward-mode differentiation. Interactions with objects whose properties are not being differentiated do not trigger any additional computation compared to a standard path tracer.

Unbiased solutions. When the scene involves very long light paths and many differentiable objects, it might therefore be preferable to precompute a data structure that enables cheap approximate queries of L_i. For instance, a path guiding technique [139] could be used to accelerate rendering in the primal phase. The resulting spatio-directional tree storing an interpolant of L_i could subsequently be used to perform radiance queries in the adjoint phase.

However, the best solution is likely the *path replay backpropagation* algorithm of Vicini et al. [21], which builds on radiative backpropagation. By inserting an additional forward rendering pass before the adjoint, storing the resulting per-ray radiance, and *replaying* the same paths in the adjoint phase, the L_i term can be recomputed for each path suffix at minimal cost and in linear time. The algorithm is described in more detail in Section 5.2.2.

Biased solutions. We now turn to a counter-intuitive finding—that a naïve formulation with biased gradients could be preferable! We propose two approximations: the first replaces the effective emission

$$\mathbf{Q}(\mathbf{x},\boldsymbol{\omega}) = \partial_\theta L_e(\mathbf{x},\boldsymbol{\omega}) + \int_{S^2} L_i(\mathbf{x},\boldsymbol{\omega}')\,\partial_\theta f_s(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}'^{\perp},$$

by a simplified expression

$$\mathbf{Q}_{\text{approx}}(\mathbf{x},\boldsymbol{\omega}) := \partial_\theta L_e(\mathbf{x},\boldsymbol{\omega}) + \int_{S^2} \partial_\theta f_s(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\omega}')\,\mathrm{d}\boldsymbol{\omega}'^{\perp},$$

where we have substituted the primal radiance term L_i with the value 1. In this case, gradients of individual material parameters often record the correct sign (namely, whether the parameter should increase or decrease), but their magnitude will generally be incorrect[8]. This change was originally motivated from an efficiency perspective: using $\mathbf{Q}_{\text{approx}}$, radiative backpropagation no longer requires recursive invocations of the primal integrator, and the quadratic time complexity thus becomes linear. This is also slightly cheaper than path replay backpropagation, since no additional forward pass is needed. Surprisingly, we found in our experiments that biased gradients are not only faster to compute, but that they can paradoxically lead to improved convergence per iteration even when accounting for their different overall scale.

To try to understand potential reasons for this effect, we refer to a study of sign-based gradient descent techniques by Balles and Hennig [140]. Comparing SGD to stochastic sign descent, which *only uses the sign* of computed gradients, the authors report: "On the well-conditioned problem, gradient descent vastly outperforms the sign-based method in the noise-free case, but the difference is evened out when noise is added." In our case, $\mathbf{Q}_{\text{approx}}$ removes a significant source of variance in the radiative backpropagation procedure, which we believe to be responsible for this counter-intuitive improvement. In the remainder of this chapter, results using this simplification are labeled *Biased (I)*.

Our second approximation builds on the observation that the adjoint phase computes many quantities found in normal path tracers: samples from the BSDF and direct illumination strategies, MIS weights, etc. At a negligible additional cost, we can therefore render an image of the scene *while propagating gradients*. This suggests the following iterative scheme with a joint primal and adjoint phase:

```
y = f(θ)
while not converged:
    δ_y = J_g y
    # Joint primal and adjoint phase
    y, δ_θ = radiative_backprop(θ, δy)
    # .. gradient step ..
```

The above iteration now contains an intentional "off-by-1 error": iteration i propagates the adjoint rendering from iteration $i-1$ through the Jacobian of iteration i.

$$\boldsymbol{\delta}_{\theta}^{(i)} = \mathbf{J}_f^T\big(\boldsymbol{\theta}^{(i)}\big)\, \boldsymbol{\delta}_{\mathbf{y}}^{(i-1)} \tag{4.2}$$

[8]The original article claimed that the correct sign is always preserved, which is not correct in general, see [21, Section 3.2].

which can be a good approximation if the Jacobian changes slowly from iteration to iteration. In the context of neural networks, this optimization is known as *pipelining* [141]. In the remainder of this chapter, results using pipelining are labeled *Biased (II)*, and results that also use $\mathbf{Q}_{\text{approx}}$ are labeled *Biased (I + II)*.

Discussion. The surprisingly good performance of the biased variants described above elicits new questions: where exactly does the balance between bias and variance lie in the context of Monte Carlo-based inverse problems? Is this tradeoff highly dependent on the scene, or can general results be derived? Optimization procedures are highly flexible and could be designed to make use of biased, but cheap gradients in early iterations, progressively switching to unbiased, but costlier estimates in the final stages of convergence. These modifications could also be leveraged in future work as a tool against suboptimal local minima.

4.4 Evaluation

We now verify the correctness of our method and apply it to several inverse rendering settings. We compare to the automatic differentiation-based method of Chapter 3, as implemented in Mitsuba 2, which will be presented in Chapter 7.

4.4.1 Validation

To test the correctness of the computed gradients, we selected several simple cases (diffuse RGB coefficients, heterogeneous medium density, diffuse texture), and compare the gradients of these parameters with respect to the loss generated by our method, Mitsuba 2's AD backend, as well as finite differences. We generally find good agreement with a small amount of residual noise, as shown in Figure 4.3.

4.4.2 Performance

To quantify the effect of approximate gradients ("biased I") and pipelining ("biased II") optimizations, we measure the runtime of the primal, adjoint or combined primal / adjoint phases in three types of scenes. The breakdown is plotted in Figure 4.4. Given a scene where all interactions involve differentiable parameters, approximate gradients will help most since they eliminate a component of the runtime that is quadratic in path

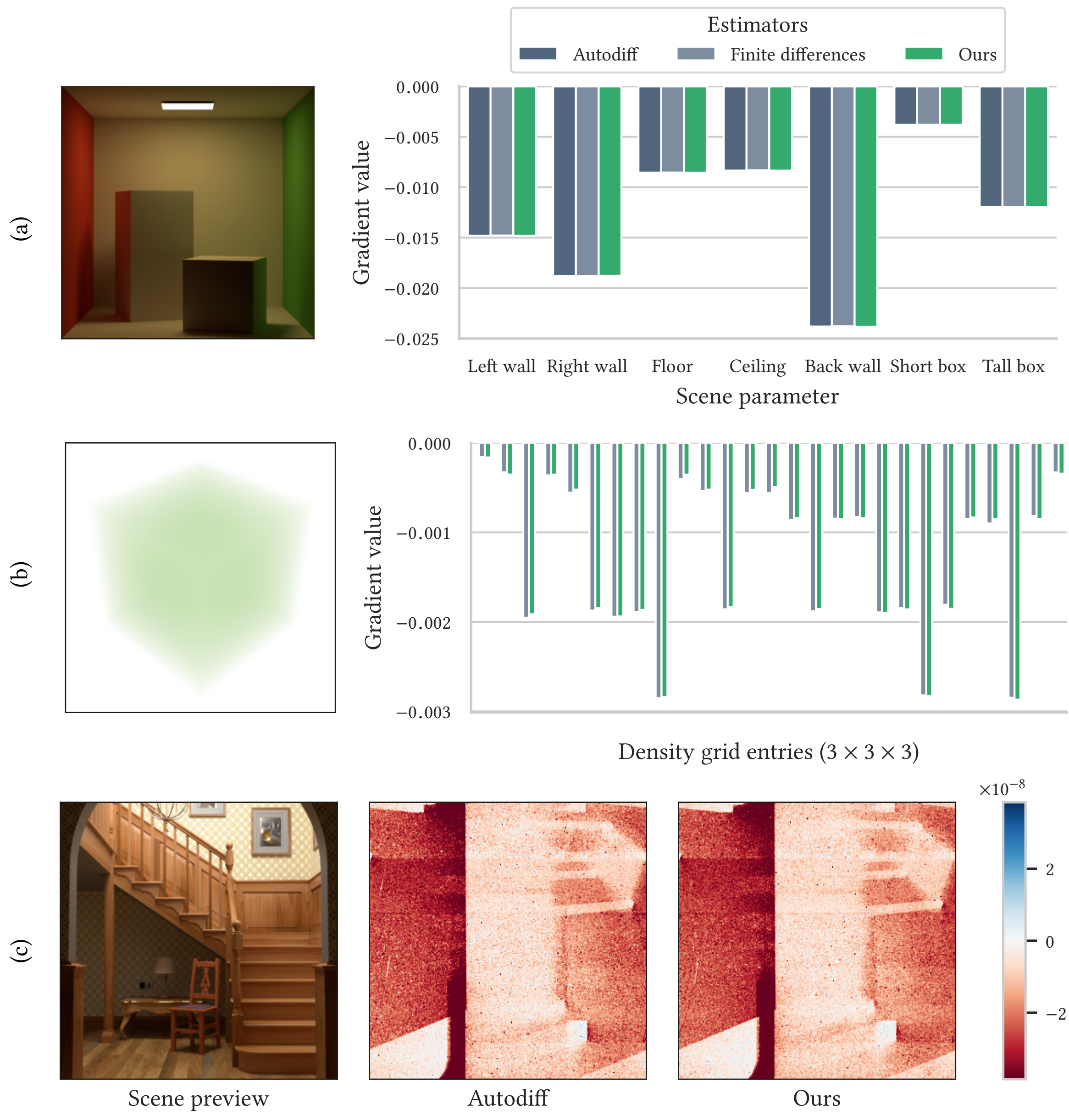
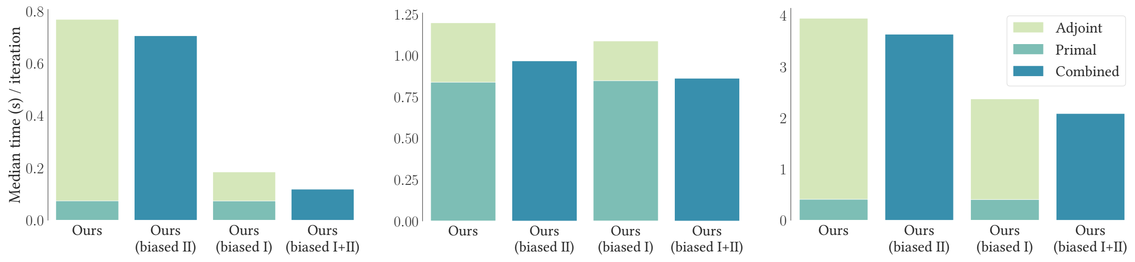


Figure 4.3: Validation of gradients comparing radiative backpropagation to Mitsuba 2's automatic differentiation mode, and finite differences when practical. We plot the partial derivative of the loss w.r.t. each parameter. (**a**) Gradients of the RGB colors of objects in the Cornell box sceme. (**b**) Derivatives with respect to extinction in a $3 \times 3 \times 3$ heterogeneous volume lit by an environment map. (**c**) Gradients with respect to the diffuse albedo texture of the wooden floor in the Staircase scene. In all cases, we find close agreement.

length. On the other hand, scenes with costly primal rendering phases may benefit most from pipelining.

More extensive performance evaluation is postponed to Chapter 8, where we expand on important aspects of the implementation.



(a) Cornell box (all materials differentiable).

(b) Figure 4.1 scene, optimizing the globe only.

(c) Smoke optimization application of Figure 4.7.

Figure 4.4: We break down the running time of radiative backpropagation into primal and adjoint phases. Using biased gradients ("biased I") drastically reduces the cost of the adjoint phase when many differentiable objects are present in the scene (**a** & **c**), while pipelining ("biased II") combines the primal and adjoint phases into a single step (**b**).

4.4.3 Texture optimization

The following experiments focus on the solution of inverse problems using gradient-based optimization. We present results for unbiased radiative backpropagation and the two approximations of Section 4.3.8: biased gradients ("biased I"), pipelining ("biased II"), and both ("biased I + II"). As before, the main focus of our work is on efficient differentiation rather than solving specific optimization problems. Additional techniques such as multi-resolution, multi-view and hyper-parameter adjustments are orthogonal to our method and could be used to further improve convergence.

Figure 4.1 showcases the reconstruction of the texture of a globe encased in a curved sheet glass modeled using two interfaces. This scene involves fine detail, and the underlying simulation accounts for paths with up to 24 scattering events including specular-diffuse-specular interreflection. In such a setting where high resolution and sample count are both required, the AD-based approach of Mitsuba 2 rapidly exhausts the available GPU memory. As a consequence, rendering and differentiation must be split in multiple smaller passes, which negatively impacts the overall running time.

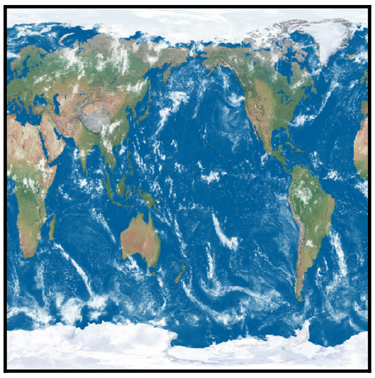
We use a simple L_2 objective function, adding a total variations regularizer to promote smoothness of the texture. Note that any differentiable loss can be used in combination with radiative backpropagation, including more complex ones built from convolutional neural networks. We render each iteration at a resolution of 1280×720 pixels and double the sample count every 40 iterations starting at 4 samples per pixel. Figure 4.5 compares the convergence of all methods, both with respect to iteration count and runtime. The optimized texture is displayed at equal time, and corresponding renderings are shown in Figure 4.1. In principle, there should be no major difference in convergence per iteration when comparing gradients that are obtained using different techniques. Surprisingly, we observe that biased gradients significantly improve convergence although they are clearly “less correct”. We believe that the primal radiance estimates performed during our method’s adjoint phase introduce large amounts of variance into gradient estimates that impede convergence. Primal radiance estimates are also implicitly used when a complete simulation is differentiated using AD, as is the case in the current implementation of differentiable rendering in Mitsuba 2. By removing this source of variance, faster progress can be achieved.

4.4.4 Volume optimization

Next, we carry out two inverse volume rendering optimizations. These experiments were originally designed by Delio Vicini to evaluate the differentiable rendering features of Mitsuba 2 [14], and we reproduce them here for comparison. The former optimizes the spatially-varying albedo of a homogeneous volume to match the appearance of a specified texture, see Figure 4.6. This is useful in the context of 3D printing, where the scattering of the material leads to blurring and a loss of contrast. Optimization can be used to determine which color should be used at each point of the medium to maximize contrast and fidelity. This problem was studied in depth in previous work [142, 143], and we merely use it here as an illustrative example.

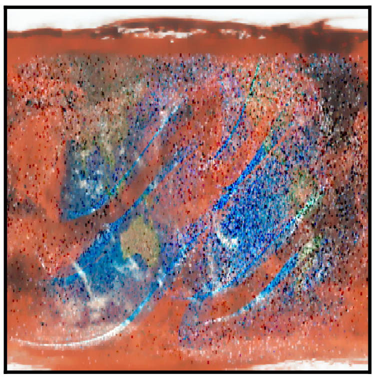
The second problem reconstructs density values σ_t of a high-resolution smoke plume from an image of the volume itself, shown in Figure 4.7. Inverting such complex transport effects would be extremely challenging without differentiable rendering.

In both cases, the AD-based baseline must maintain the transcript of the entire volumetric transport simulation before being able to perform a reverse-mode traversal. This transcript becomes staggeringly large for any non-trivial volume resolution due to a long sequence of trilinear lookups. Despite being rendered at 256×256 resolution, these examples exhaust GPU memory with as few as 1 sample per pixel. In contrast, radiative backpropagation uses a minimal amount of memory and never requires multiple passes. Its memory usage is essentially independent of the volume resolution, as only the volume and its gradients must be stored.

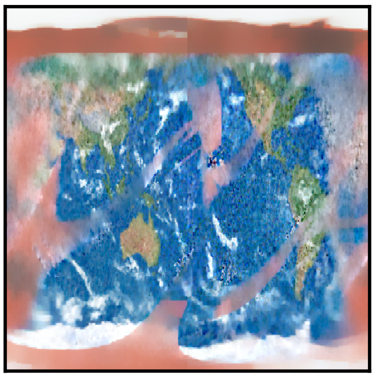


(a) Reference

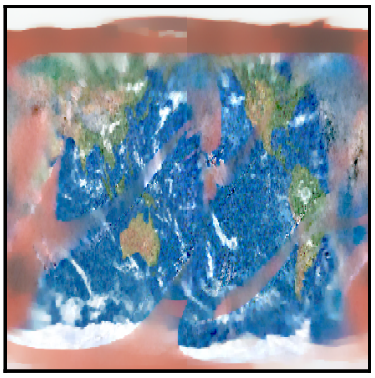
(b) Initial state



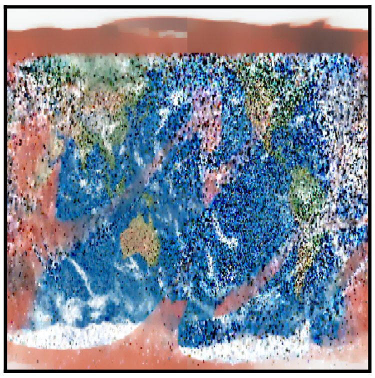
(c) Mitsuba 2 (AD)



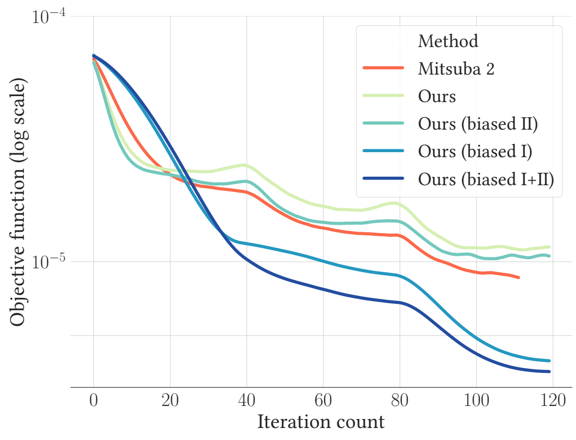
(d) Ours (biased I)



(e) Ours (biased I+II)



(f) Ours



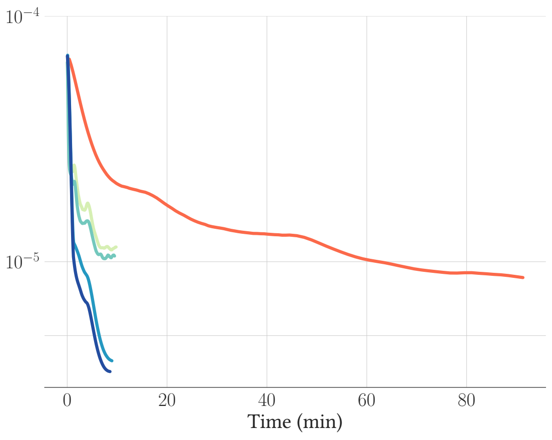
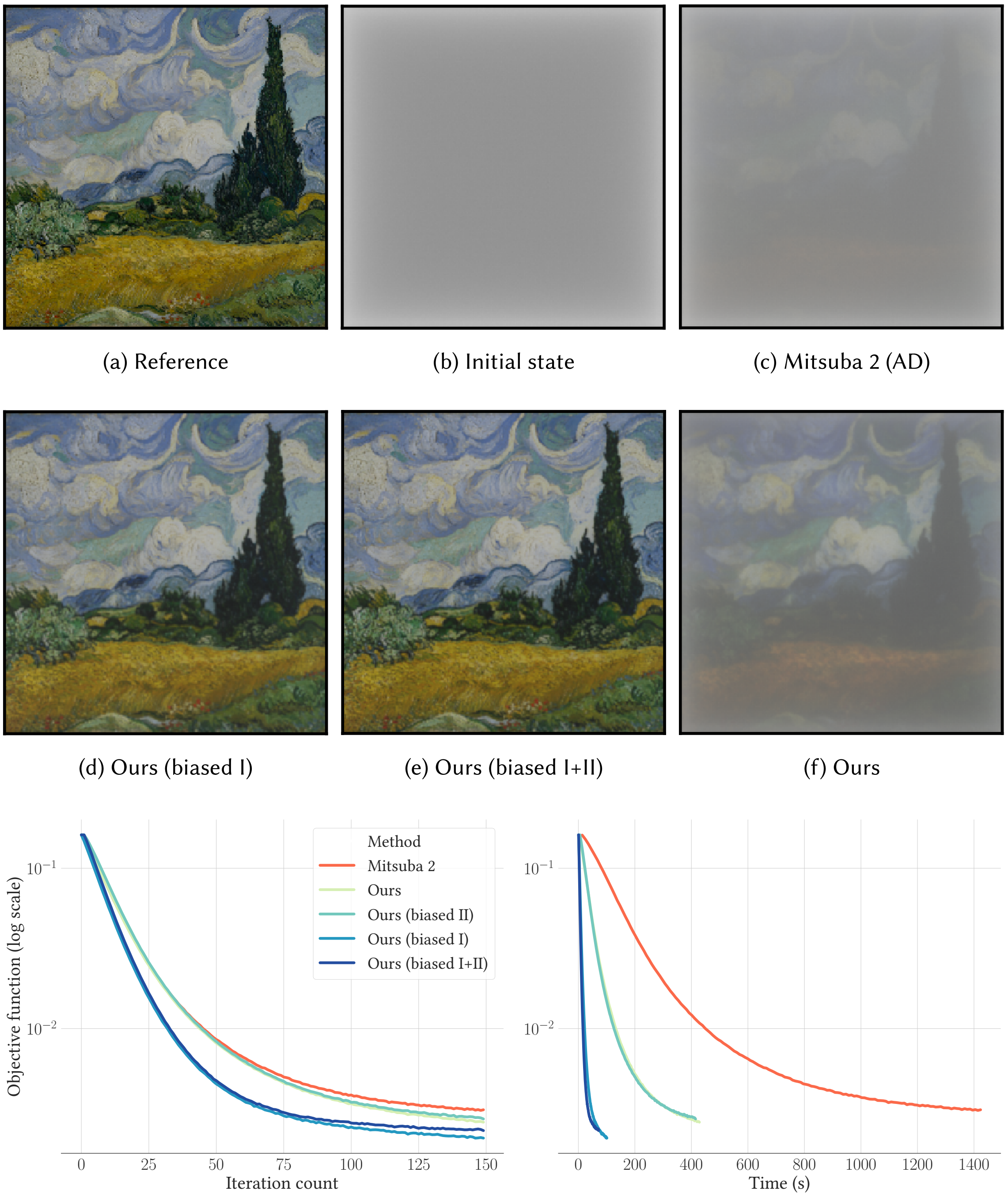


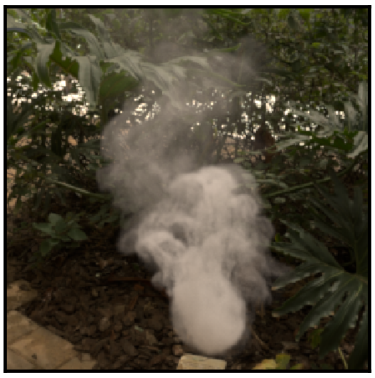
Figure 4.5: Recovery of the diffuse globe texture shown in Figure 4.1 from a single reference image. Despite indirect observation via multiple refractions and reflections in a scene with complex transport, our method is able to closely match the reference image in less than ten minutes (Figure 4.1 insets). At the top, we show the reconstructed textures after 8.5 minutes. Regions that are not visible (even indirectly) remain close to the initial state. A variant of our technique ("Biased I") removes a significant source of variance by approximating incident radiance with a constant, improving convergence at equal iteration count.



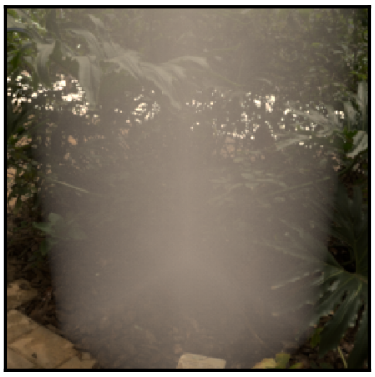
(a) Reference (b) Initial state (c) Mitsuba 2 (AD)

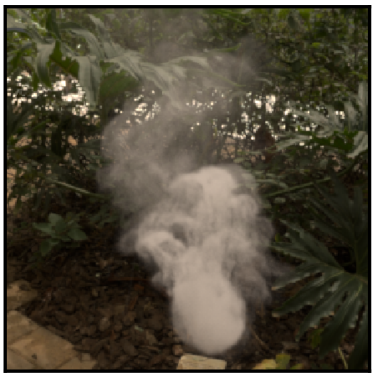
(d) Ours (biased I) (e) Ours (biased I+II) (f) Ours

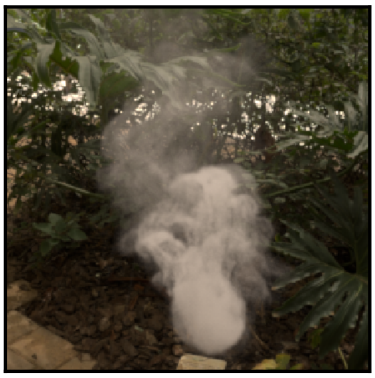
Figure 4.6: We reproduce the albedo optimization experiment of Nimier-David et al. [14], in which the spatially-varying albedo of a homogeneous scattering slab must be modified to match the appearance of a diffuse texture while accounting for subsurface scattering. Starting from a constant gray slab, our method achieves convergence after a single minute of optimization. The resulting media are shown at the top at equal time (74 seconds). In this example, using biased gradients (“Biased I”) avoids costly recursive estimation of incident radiance, which dramatically reduces the runtime cost per iteration.

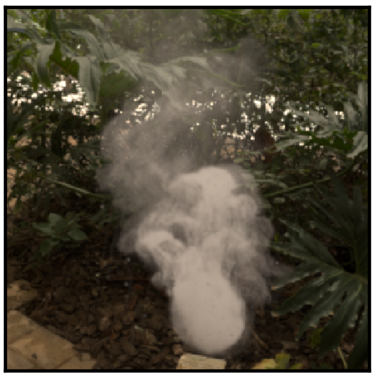

(a) Reference

(b) Initial state


(c) Mitsuba 2 (AD)


(d) Ours (biased I)


(e) Ours (biased I+II)


(f) Ours

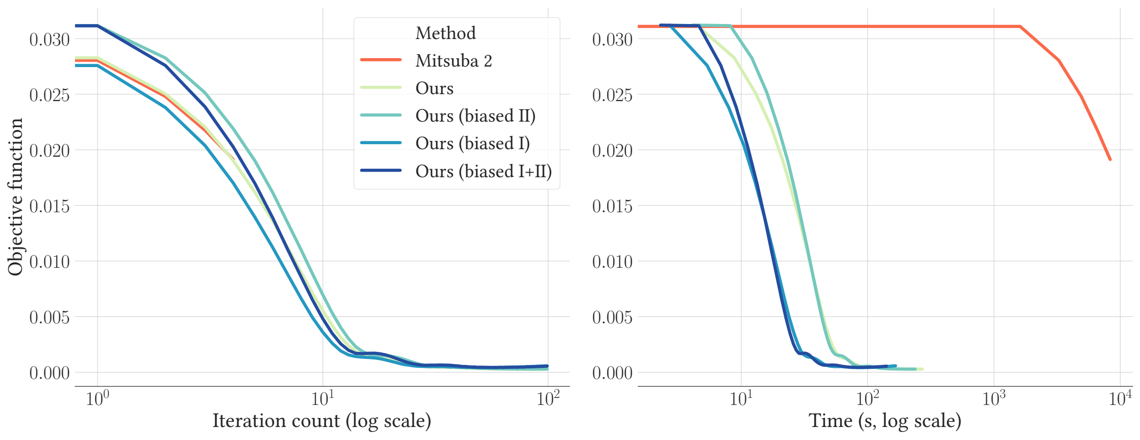


Figure 4.7: We also reproduce the volume density optimization experiment of Nimier-David et al. [14]. In this example, Mitsuba 2's AD-based approach reaches its limits, as the size of the transcript it must store immediately fills the GPU memory, even for low-resolution 128×128 renderings. Insets at the top show convergence at equal time (2.5 minutes), or at the end of the first iteration in case of Mitsuba 2 (26 minutes).

4.5 Conclusion

Differentiable rendering has clear potential as a tool for solving inverse problems in a variety of scientific disciplines. However, previous approaches to differentiable rendering remain hamstrung by a fundamental performance and scalability bottleneck: reverse-mode automatic differentiation produces vast amounts of intermediate state that rapidly exhausts the memory of even the largest available computing platforms. This currently limits the scope of this technology to relatively simple scenes rendered at low resolutions.

Our method conclusively addresses these limitations. Our main contribution is a new approach to differentiable simulation of light that does not require a transcript of intermediate state, thus avoiding burdensome storage overheads and improving performance by up to three orders of magnitude. This is remarkable because transcript-less differentiation of simulation code was previously only possible in rare cases, one famous example being the adjoint sensitivity method, which relies on the ability to reverse the flow of time.

We show that a different kind of adjoint can also be constructed for steady-state light transport simulations that lack a time dimension. In our case, the adjoint phase emits two differential quantities from sensors and objects that propagate towards each other akin to ordinary light. Their eventual encounter yields a gradient measurement that turns the chain rule from a discrete sum over partial derivatives into a continuous integral on ray space. Another contribution of our work is that it casts the adjoint phase into familiar light transport terms, enabling the application of a large body of prior work on sampling transport integrals.

One limitation is that the unbiased version of our algorithm has a time complexity that is quadratic in the path length which could become prohibitive (*e.g.* in highly scattering participating media with light paths involving thousands of interactions). We propose a biased variant that addresses this performance concern and, oddly, appears to generally improve convergence per iteration.

Our approach admits an efficient GPU implementation using standard toolkits for megakernel-based rendering. In Chapter 8, we discuss this implementation in detail and demonstrate the automated creation of adjoint shaders building on Mitsuba 2's tracing JIT compiler.

Impact. Vicini et al. [21] have proposed *path replay backpropagation* (PRB), which builds on radiative backpropagation to bring the complexity back to linear time. Memory usage remains constant w.r.t. to the path length. PRB relies on the deterministic nature of pseudo-random number generators to first compute the incident radiance along the paths, before *replaying* them to carry out the backpropagation. Furthermore, the algorithm adds support for directional derivatives, enabling *e.g.* the optimization of surface normals.

Furthermore, Zeltner et al. [12] show that radiative backpropagation can be extended to include visibility derivatives by combining the algorithm with the aforementioned reparametrization-based technique.

Radiative backpropagation has already been leveraged in subsequent work. For example, Nindel et al. [130] use our method to optimize parameters of ink mixtures in the context of 3D printing to finely control the final appearance.

Future work. Our method currently resembles an ordinary path tracer with multiple importance sampling, and smarter adjoint-specific sampling strategies could be beneficial. Chapter 5 explores such a strategy dedicated to the volumetric case. Finally, our work focused specifically on the evaluation of gradients, but the underlying optimization problem might be challenging even if perfect gradients were freely available. Further work on techniques to regularize the energy landscape of differentiable rendering is therefore imperative. Chapter 11 proposes heuristics in this direction in the context of indoor room reconstruction from real data.

5 | Unbiased Inverse Volume Rendering with Differential Trackers

Relevant background: Sections 2.1.3, 2.1.7 and 2.2.3.

In Chapter 4, we introduced radiative backpropagation, and adjoint method for efficient gradient estimation. One of its advantages is that we can freely apply different importance sampling techniques to the primal and adjoint problems. In this chapter, we identify a specific case—in-scattering gradients for differentiable volume rendering—which requires a dedicated sampling scheme to avoid bias and high variance. To this end, we propose differential ratio tracking and demonstrate its effectiveness in eliminating bias and variance concerns.

5.1 Introduction

Volumetric inverse rendering approaches are particularly attractive due to their flexibility in representing both hard and soft surfaces, their inherent smoothness, as well as their invariance to topological changes. Nonetheless, the efficient evaluation of radiative transfer and its derivatives—as required in a gradient descent algorithm—remains difficult. Current methods generally follow two strategies to cope with this difficulty: **(i)** simplifying the model, such as by only permitting emission and absorption [118] or limiting the permissible materials and illumination [144], and **(ii)** developing more advanced algorithms for efficiently differentiating the radiative transfer equation in its general form [15, 21, 109]. This chapter contributes to the latter direction, while Chapter 10 discusses the usage of simplified volumetric models in the context of physically based inverse volume rendering.

First, we follow the general observation that forward-rendering strategies are not necessarily suitable for differential rendering [12, 115], showing in this instance that standard volumetric free-flight sampling yields biased and high-variance gradients. This can be explained intuitively: free-flight sampling places samples on the portions of the volume contributing most to its appearance (formally: proportionally to the product of transmittance and density). However, even in empty space, gradients may have non-trivial magnitude, *e.g.* to reconstruct an object where previously there was just air. In Section 5.4, we show that placing samples in *all* visible regions (proportionally to just

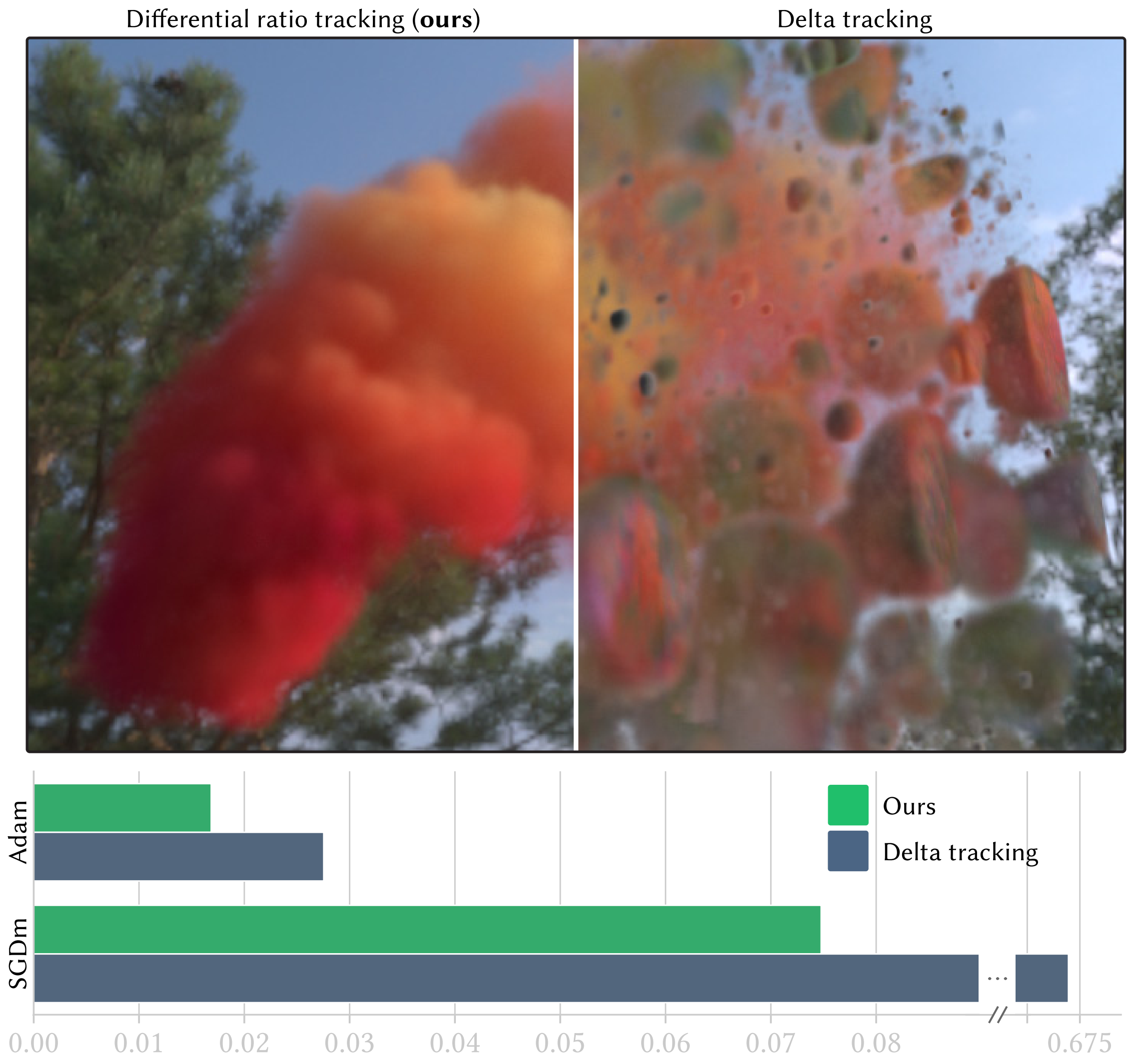
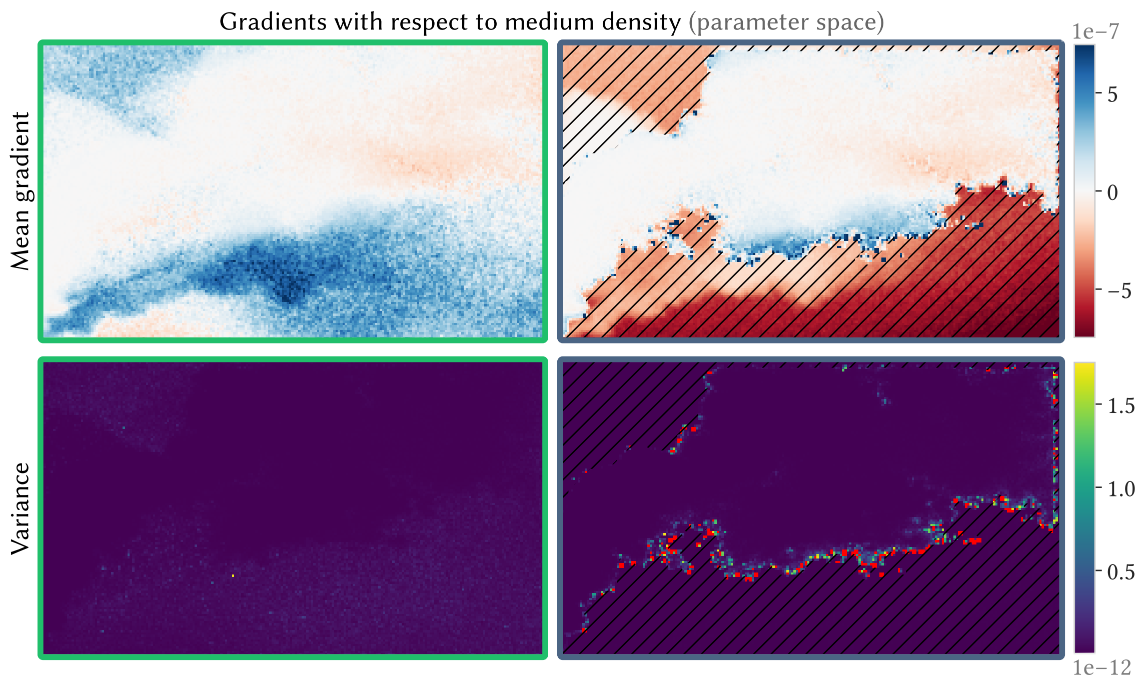


Figure 5.1: We demonstrate the high-quality reconstruction of volumetric scattering parameters from RGB images with known camera poses and lighting. This is enabled by our novel differential ratio tracking formulation, which yields unbiased, low-variance gradients of the radiative transfer equation that can be directly used for optimization. In the chart (bottom), we report the improvements in reconstruction error for stochastic gradient descent with momentum (SGDm) as well as Adam. Using aggressive step size reduction, the Adam optimizer limits the impact of large gradient outliers, though our unbiased gradients lead to the lowest reconstruction error with either optimizer.



(a) Differential ratio tracking (ours) (b) Delta tracking

Figure 5.2: Continued from Figure 5.1. Traditional free-flight sampling—*e.g.* by delta tracking—while effective at low-variance *primal* rendering, exhibits bias and high variance in gradient estimation with respect to medium density (**b**), which negatively affects optimization. Our method (**a**) fully eliminates bias and variance concerns. Gradient mean and variance values are shown for slice $z = 64$ of the $256\times128\times128$ parameter space.

transmittance) significantly reduces the variance of these types of gradients (Figure 5.2). Based on this insight, we propose a novel sampling scheme, *differential ratio tracking*, that combines (residual) ratio tracking and reservoir sampling to sample distances proportionally to transmittance.

We use an additional weighted reservoir to integrate differential ratio tracking into the linear-time path replay backpropagation algorithm [21], leading to better-behaved gradient descent optimization and lower reconstruction error with limited overhead, as shown in Figures 5.1 and 5.3. Our sampling scheme is dedicated to physically-based scattering and absorbing volumes (which are inherently relightable), as opposed to NeRF-style emissive volumes [118].

We begin by reviewing the relevant volumetric rendering theory (Section 5.2). Our differential ratio tracking strategy for low-variance gradient estimation is derived in Section 5.4. We conclude with an evaluation and discussion of our method in Section 5.5.

5.2 Background

We review the most relevant aspects of volumetric path tracing (primal rendering) and the radiative backpropagation formulation for gradient estimation (adjoint rendering).

5.2.1 Volumetric path tracing

We adopt the notation introduced in Section 2.2.3. Recall the volume rendering equation (2.22)

$$\begin{aligned} L_i(\mathbf{x}, \boldsymbol{\omega}) = &\int_0^{t_s} \mathrm{T}(t) \left[\sigma_a(t)\, L_e(t) + \sigma_s(t)\, L_s(t, \boldsymbol{\omega})\right] \mathrm{d}t \\ &+ \mathrm{T}(t_s) \left[L_e(t_s) + L_s(t_s, \boldsymbol{\omega})\right]. \end{aligned} \tag{5.1}$$

Before considering the estimation of gradients, volumetric rendering requires dedicated algorithms and sampling methods, generalizing the surface case.

Free-flight distance sampling. Estimating Equation (5.1)—as well as its null scattering form, Equation (A.1)—with Monte Carlo integration involves efficiently sampling a so-called *free-flight* distance t, which corresponds to the distance to the next light-particle interaction along the ray. This distance is distributed proportionally to $\sigma_t(t)\,\mathrm{T}(t)$, which can be sampled in closed form in homogeneous volumes. In heterogeneous volumes, however, a more advanced algorithm must be used. *Delta tracking* [145], also known as Woodcock tracking [146], *homogenizes* the medium by introducing fictitious matter such that the extinction coefficient of the volume is equal a chosen *majorant* value $\bar{\sigma}$ everywhere.

The algorithm then analytically samples a free-flight distance proportionally to $\bar{\sigma}\,\bar{\mathrm{T}}(t)$, where $\bar{\mathrm{T}}(t)$ is the homogenized transmittance. Finally, the ray is stochastically determined to have encountered a *real* particle with probability $\sigma_t(t)/\bar{\sigma}$, or a fictitious one otherwise. The algorithm repeats until a real interaction has been found; the resulting distance t has been shown to be distributed proportionally to $\sigma_t(t)\,\mathrm{T}(t)$ as desired [146]. The choice of majorant $\bar{\sigma}$ affects the efficiency: the more closely it bounds σ_t, the smaller the number of fictitious interactions that are processed—but the higher the variance.

Transmittance estimation. A related operation in volumetric light transport is estimating the transmittance $\mathrm{T}(t)$ along a ray segment: the fraction of light traveling a distance t without being absorbed or scattered away. Transmittance can be estimated in a number of ways, such as by marching along the ray in fixed intervals and correcting for bias [147], and, most relevant for us, by free-flight sampling and thus delta tracking. Delta tracking can be thought of as a *binary* estimate of $\mathrm{T}(t)$. The estimate is one (all light is transmitted) if a sampled free-flight distance is larger than t and zero (no light is transmitted) otherwise. This interpretation can be generalized to lower-variance, non-binary estimates of the transmittance by keeping track of the proportion of real vs. fictitious matter $\sigma_t(t)/\bar{\sigma}$ at each step; this is called *ratio tracking* [148]. Our method combines ratio tracking and reservoir sampling to sample proportionally to $\mathrm{T}(t)$ only rather than $\sigma_t(t)\,\mathrm{T}(t)$. We will show this density to be desirable for unbiased, low-variance gradient estimation in (near-)empty regions of space; illustrated in Figure 5.4.

5.2.2 Path replay backpropagation

In Chapter 4, we introduced *radiative backpropagation*, an adjoint method for efficient gradient estimation. As presented, the unbiased variant of radiative backpropagation requires starting recursive paths at each interaction with differentiable objects in order to obtain an estimate of the incident radiance L_i. This raises the worst-case complexity to $\mathcal{O}(n^2)$, where n is the number of path vertices. *Path replay backpropagation* [21] leverages the deterministic nature of pseudo-random number generators to bring complexity back to linear time and constant memory.

Our method does not depend on either radiative or path replay backpropagation. However, these adjoint methods make it possible to efficiently estimate gradients for millions of parameters in parallel, including in the presence of high-order scattering. Furthermore, their formulation makes it straightforward to apply separate specialized sampling strategies to the *primal-* (forward rendering of the scene) and *adjoint* problems (estimating gradients w.r.t. scene parameters). It is thus desirable to ensure that our proposed algorithm is compatible with them.

Path replay backpropagation consists of three steps:

(I) A forward (primal) rendering of the current state of the scene is performed and used to evaluate the objective function.

(II) A second, uncorrelated[1] primal rendering pass estimates and stores the radiance L_i^{II} for each path.

[1]The need for an uncorrelated rendering pass is discussed in Section 3.2.

(III) An adjoint rendering pass *replays* the paths traced in step (II) using the same random number generator seed. The stored incident radiance values L_i^{II} are used to weight gradients, without the need to trace a recursive path.

In the context of this chapter, it is important to note that steps (II) and (III) must use the same paths, and hence the *same sampling methods* to construct them. We must take care to make any new sampling technique compatible with this constraint in order to preserve the linear runtime (Section 5.4.5).

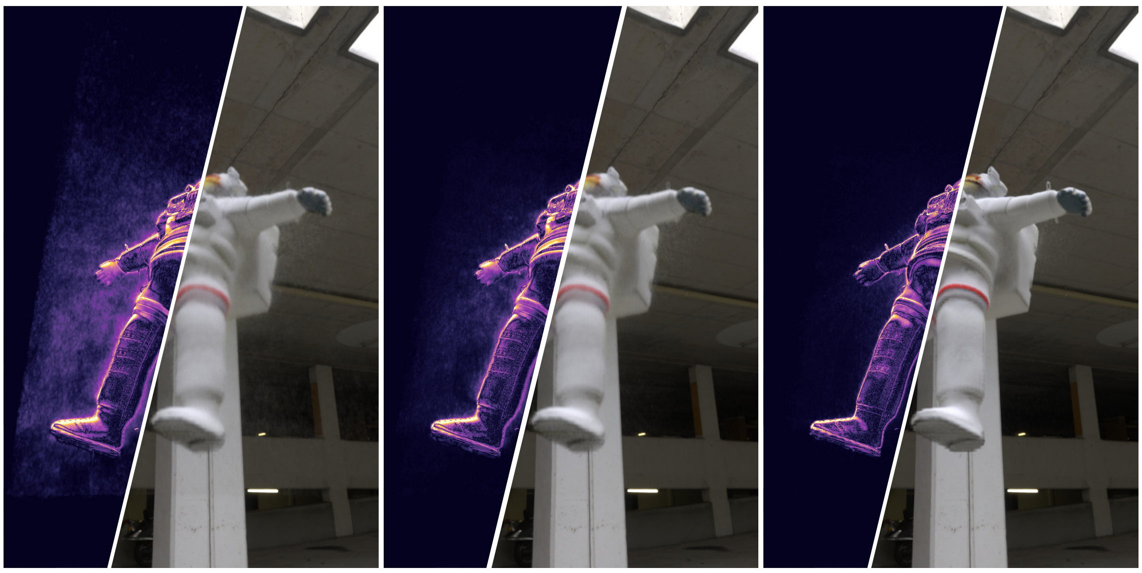

(a) Delta Tracking (b) Ours (constant init.) (c) Ours (emissive init.)

Figure 5.3: Optimizing a scattering volume to reproduce an object from 64 reference images. The false-color visualization indicates the absolute difference between the final result and the reference image. **(a)** Using the same free-flight sampling technique to estimate both the primal image and the gradients results in biased and high-variance gradients (Section 5.3.2), hindering convergence in empty or low-density regions. **(b)** Using our novel differential ratio tracking technique (Section 5.4.1) significantly reduces gradient variance, leading to a better reconstruction. **(c)** We further propose to initialize the scattering volume optimization from the result of a nonphysical emissive volume optimization in the style of NeRF [118] (Section 10.1). This helps overcome local minima, greatly improving the sharpness of the final model.

5.3 Issues with free-flight based gradient estimation

We now turn to the problem of efficiently estimating gradients with respect to scene parameters in the presence of heterogeneous media.

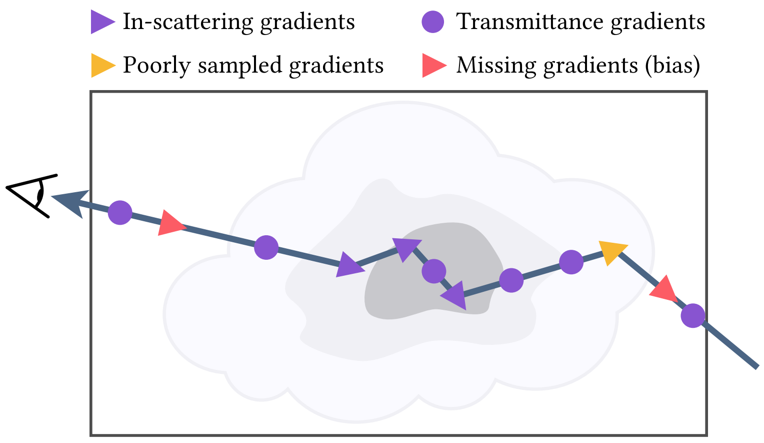


Figure 5.4: Forward rendering of volumes involves sampling free-flight distances t proportionally to the density-weighted transmittance $\sigma_t(t)\,\mathrm{T}(t)$. Directly applying forward sampling techniques in the context of gradient estimation (the adjoint) leads to biased and high-variance gradients. In particular, gradients due to in-scattering are poorly sampled where density is low and never sampled where it is zero. We propose a tailored importance sampling scheme for the adjoint, which resolves both bias and variance concerns.

5.3.1 Differentiating the radiative transfer equation

Reconstructing media that are both reflective and emissive introduces ambiguities that are challenging to resolve using only image-based observations. Since the appearance of real-world objects is usually determined by their scattering properties rather than internal emission, we restrict our goal to reconstructing non-emissive volumes that obtain their color from scattering and absorption; we set $L_e = 0$ within the medium. Using the $\boldsymbol{\theta} = (\sigma_t, \alpha)$ parametrization of the medium, the radiative transfer equation (5.1) simplifies to:

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = \int_0^{t_s} \mathrm{T}(t)\,\sigma_t(t)\,\alpha(t)\,L_s(t, \boldsymbol{\omega})\,\mathrm{d}t + \mathrm{T}(t_s)\,\big[L_e(t_s) + L_s(t_s, \boldsymbol{\omega})\big]\,, \tag{5.2}$$

where t_s denotes the position of the nearest surface (or volume bounding box) in the direction of the ray.

For a given ray traversing the medium, the radiance derivatives w.r.t. volume parameters $\boldsymbol{\theta}$ are as follows. We separate and rearrange the terms to explain their individual roles, as well as to map more directly to the sampling algorithms. We omit the dependency on $\boldsymbol{\omega}$ for conciseness.

The first term captures how the *in-scattered* radiance can increase due to a local density (σ_t) or albedo (α) increase:

$$\partial_\theta L_i(\mathbf{x}) = \int_0^{t_s} T(t)\ \partial_\theta\big[\sigma_t(t)\ \alpha(t)\big] L_s(t)\,\mathrm{d}t\ \cdots \tag{5.3}$$

The second term describes how a density increase at an earlier position t' will attenuate the local contribution, whether from a medium interaction (first row) or a surface (second row):

$$\begin{aligned}\cdots + \int_0^{t_s} T(t)\,\sigma_t(t)\,\alpha(t)\ &\left[\int_0^{t} -\partial_\theta\sigma_t(t')\,\mathrm{d}t'\right] L_s(t)\,\mathrm{d}t \\ + T(t_s)&\left[\int_0^{t_s} -\partial_\theta\sigma_t(t')\,\mathrm{d}t'\right]\big[L_e(t_s) + L_s(t_s)\big]\ \cdots\end{aligned} \tag{5.4}$$

Finally, the third term captures changes in incident radiance after a scattering event at t in the medium or at t_s on a surface. Changes later along the path are weighted by the transmittance of the current segment and the scattering coefficient of the current event:

$$\begin{aligned}\cdots + &\int_0^{t_s} T(t)\,\sigma_t(t)\,\alpha(t)\,\partial_\theta L_s(t)\,\mathrm{d}t \\ + &T(t_s)\left[\partial_\theta L_e(t_s) + \partial_\theta L_s(t_s)\right].\end{aligned} \tag{5.5}$$

5.3.2 Free-flight based gradient estimators

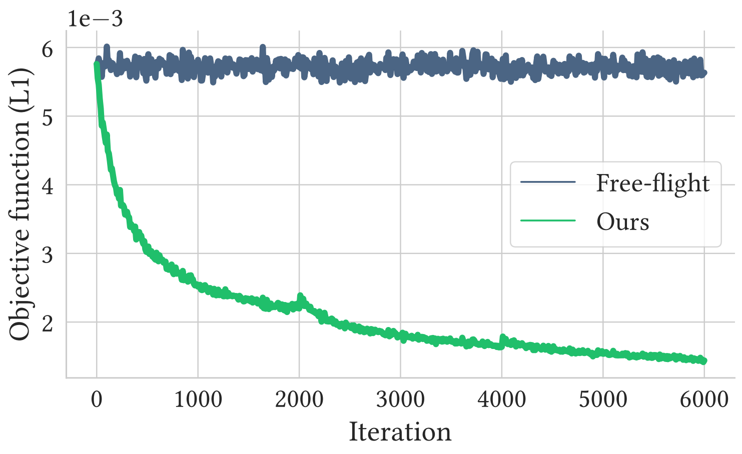
Standard free-flight distance sampling proportionally to $\sigma_t(t)\,T(t)$, such as by delta tracking, is well suited for the primal problem of Equation (5.2): the probability density function cancels most factors. However, using free-flight sampling to estimate the adjoint is problematic, as we will show below. Note its usage can be inadvertent, *e.g.* when applying automatic differentiation to a volumetric path tracer.

Using free-flight importance sampling to sample the adjoint integrals (5.3)–(5.5) yields the following estimators. A valid free-flight distance $t < t_s$ is sampled within the medium with probability $\sigma_t(t)\,\mathrm{T}(t)$, resulting in a Monte Carlo sample with contribution

$$\langle \partial L_1^{\mathrm{DT}} \rangle = \frac{\partial_\theta\big[\sigma_t(t)\ \alpha(t)\big]}{\sigma_t(t)}\,L_s(t) + \alpha(t)\ \partial_\theta L_s(t)\,. \tag{5.6}$$

Otherwise, if $t \geq t_s$, the surface at t_s is sampled with probability $\mathrm{T}(t_s)$, contributing

$$\langle \partial L_2^{\mathrm{DT}} \rangle = \ \partial_\theta L_e(t_s) + \partial_\theta L_s(t_s)\,. \tag{5.7}$$



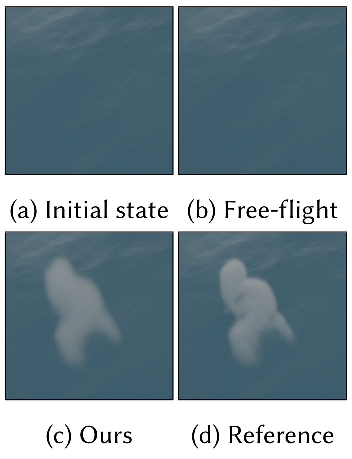


Figure 5.5: In this illustrative example, the medium density σ_t is optimized to match 32 reference images of a cloud **(d)**. The initial state **(a)** is set to $\sigma_t = 0$. This represents a worst-case scenario for the free-flight based gradient estimator **(b)**: its bias prevents it from escaping the initial state. Indeed, in-scattering gradients are missing where $\sigma_t = 0$ and transmittance gradients point in the direction opposite to the solution (*i.e.* further decreasing σ_t to increase brightness, which is impossible). Our method correctly estimates in-scattering gradients even in empty space, which allows it to converge to a good solution **(c)**.

In either case, transmittance gradients are accumulated at locations t' along the ray segment $(0, \min(t, t_s))$ corresponding to the location of null-scattering interactions:

$$\langle \partial L_3^{\mathrm{DT}} \rangle = \begin{cases} -\partial_\theta \sigma_t(t')\, \alpha(t)\, L_s(t) & \text{if } t < t_s, \\ -\partial_\theta \sigma_t(t') \left[L_e(t_s) + L_s(t_s) \right] & \text{otherwise.} \end{cases} \tag{5.8}$$

We will refer to these estimators as "free-flight" as a shorthand for free-flight sampling-based gradient estimators. There are two issues with estimator $\langle \partial L_1^{\mathrm{DT}} \rangle$.

Bias. If the medium density $\sigma_t(t)$ is zero, estimator $\langle \partial L_1^{\mathrm{DT}} \rangle$ is never sampled, even though the integrand of Equation (5.3), responsible for in-scattering gradients, is not zero. This results in bias. Figure 5.5 illustrates this bias in a specially constructed worst-case scenario, with $\sigma_t(t)$ initialized to zero. Since the background is darker than the target image, transmittance gradients point in the wrong direction: brightening the image would require increasing transmittance by further lowering σ_t, which is not possible. Because in-scattering gradients are missing, the optimization cannot escape this initial state. Interestingly, an analogous situation arises for surface BSDF optimization when the BSDF is exactly zero. In that case, however, it is legitimate to clamp BSDFs to a small nonzero value, as truly black surfaces do not occur in normal circumstances.

Variance. Furthermore, if the density $\sigma_t(t)$ is *close to* zero, the factor $1/\sigma_t(t)$ in Equation (5.6) leads to large gradient outliers in gradients w.r.t. density σ_t. Because distances are sampled proportionally to extinction-weighted transmittance, it may appear that the low probability of sampling t where $\sigma_t(t)$ is small would compensate for the additional variance. However, Tregan et al. [149] have shown (for the case of a homogeneous slab) that the resulting variance is actually unbounded[2]. Note that this issue occurs as well when using the null-scattering integral formulation, see Appendix A. Gradients w.r.t. albedo α, however, are not affected.

The effect of these gradient outliers is illustrated on an inverse rendering problem in Figure 5.1. Reconstruction becomes impossible using standard stochastic gradient descent. Optimizers that track gradient variance via second moments, such as Adam [150], will at this point aggressively reduce the step size, enabling acceptable convergence. Reconstructions, however, remain inferior to that obtainable with the unbiased gradient estimators presented in this chapter, see Figure 5.3 and Chapter 10.

Defensive sampling. A seemingly obvious solution to the problem we have just identified would be to modify the sampling strategy to include a constant "background density" ϵ:

$$p_{\text{def}}(t) = (\sigma_t(t) + \epsilon)\, \mathrm{T}_\epsilon(t)\,, \tag{5.9}$$

which can be interpreted as a form of *defensive sampling*. If ϵ is sufficiently large, the problematic term in the adjoint would no longer produce gradient outliers:

$$\frac{1}{\sigma_t(t) + \epsilon} \leq \frac{1}{\epsilon}\,. \tag{5.10}$$

In the context of correlated sampling for perturbation analysis of reactor designs, Rief et al. [83] and Rief [151] have proposed a similar scheme, where density is added in near-empty regions. However, their solution forces scattering interaction with the added density to be fully forward scattering.

While the elimination of gradient outliers may seem like a clear advantage, we must also consider the effect of the modified density on all other terms of the adjoint. Importantly, the unmodified density allowed path throughput to stay close to one in Equations (5.6-5.7). With p_{def}, non-unit sampling weights compound over the length of the path. In a volumetric rendering context where long paths are common, this can become problematic.

[2]More precisely, the argument of Tregan et al. concerns variance due to the reciprocal event of null scattering where $\sigma_n \approx \bar{\sigma}$, which is analogous to $\sigma_t \approx 0$. This is discussed in more detail in Appendix A.

Moreover, recall that when using path replay backpropagation (Section 5.2.2), the same paths must be constructed in phases (II) and (III). This means that either the sub-optimal defensive sampling strategy must also be used to estimate the incident radiance terms in the adjoint, injecting additional variance into the gradient estimates, or defensive samples must be drawn *in addition* to the paths constructed in phases (II) and (III), increasing cost. In the following, we will take the latter approach—drawing additional samples—however using a sampling technique that is tailored to the integrand of Equation (5.3).

Alternative strategies. Adopting a ray marching-based estimator would be susceptible to the same issues, even when ignoring the bias due to the discretization of the transmittance function. Indeed, the underlying density being sampled, $\sigma_t(t)\,\mathrm{T}(t)$, is identical to the delta tracking-based estimator discussed above. Likewise, we did not find reparameterizing the volume density, *e.g.* $\sigma_t(t) = \log(\theta(t))$, to be a suitable solution—near-empty regions retain high-variance gradients, and are moreover susceptible to vanishing gradients. Usage of track-length estimators [83] could lead to different variance characteristics, and should be investigated in future work.

5.4 Differential ratio tracking

5.4.1 Unbiased estimators

We have identified the $1/\sigma_t$ term of Equation (5.6) to be the source of bias and variance. Fundamentally, the issue stems from the direct application of a primal sampling technique to the adjoint problem, where the integrand is different. This same problem was identified in other contexts by Zeltner et al. [12]. We propose replacing the problematic $\langle \partial L_1^{\mathrm{DT}} \rangle$ estimator with a simple, tailored estimator that samples distances proportionally to transmittance only.

The need to adopt a specialized sampling approach makes intuitive sense: regions of the medium with $\sigma_t \approx 0$ are only sampled rarely with primal sampling strategies (or not at all, if $\sigma_t = 0$). At the same time, if transmittance is close to 1, then a small change of σ_t can have a significant impact on the solution of Equation (5.3). These regions therefore generate gradients during the adjoint transport step and must receive sufficiently many samples to produce low-variance gradient estimates. Sampling proportionally to transmittance addresses both the bias and variance issues discovered in Section 5.3.2.

Assuming for a moment that we are able to sample $t' \sim \mathrm{T}(t')$, we replace the two summands in $\langle \partial L_1^{\mathrm{DT}} \rangle$ (Equation (5.6)) by the following two respective estimators:

$$\langle \partial L_{1a}^{\mathrm{DRT}} \rangle := L_s(t')\, \partial_\theta \big[\sigma_t(t')\, \alpha(t')\big]\,, \tag{5.11}$$

$$\langle \partial L_{1b}^{\mathrm{DRT}} \rangle := \alpha\, \partial_\theta L_s(t)\,. \tag{5.12}$$

Here, t' is used *only* to sample in-scattering gradients $\langle \partial L_{1a}^{\mathrm{DRT}} \rangle$ where the original strategy was problematic. For all other terms, $\sigma_t \mathrm{T}$ is a better suited density due to cancelling more factors, therefore we keep $t \sim \sigma_t \mathrm{T}$ in $\langle \partial L_{1b}^{\mathrm{DRT}} \rangle$ and set $\langle \partial L_2^{\mathrm{DRT}} \rangle := \langle \partial L_2^{\mathrm{DT}} \rangle$ as well as $\langle \partial L_3^{\mathrm{DRT}} \rangle := \langle \partial L_3^{\mathrm{DT}} \rangle$ to estimate the remaining terms.

Readers familiar with path replay backpropagation may notice a potential drawback in Equation (5.11): because t' is sampled from a different strategy than the one used to construct the path, the path replay algorithm cannot efficiently produce an estimate of incident illumination L_i at t'. In the worst case, we may have to start a recursive path at each sampled point t', raising the time complexity to $O(n^2)$. We will return to this issue in Section 5.4.5.

5.4.2 Sampling proportionally to transmittance

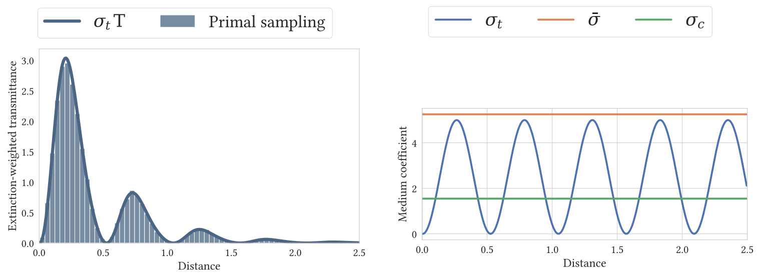
We now turn to the derivation of our transmittance importance sampling technique. To this end, we build over the family of delta- [145, 146] and ratio trackers [148].

Target distribution. Let us write down the density $p_{\mathrm{T}}(t)$ of this hypothetical transmittance based sampling strategy. Because T is not a normalized density on its own, we must apply a normalization constant defined in terms of an integral:

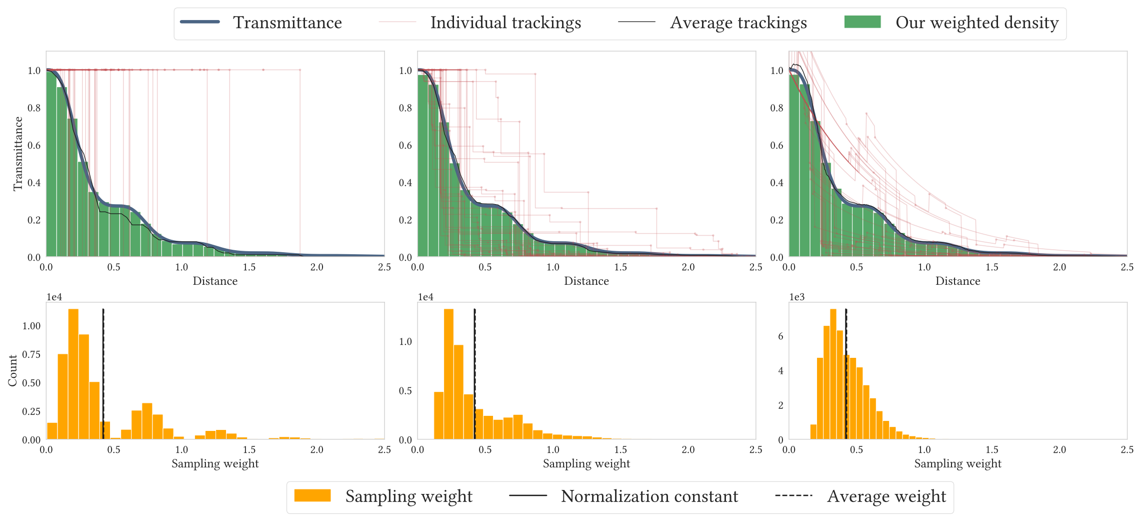
$$p_{\mathrm{T}}(t) = \frac{\mathrm{T}(t)}{\int_0^{t_s} \mathrm{T}(t')\,\mathrm{d}t'} = \frac{\mathrm{T}(t)}{C}\,. \tag{5.13}$$

The unbiased estimation of such an integral in the reciprocal presents a challenge. Moreover, T is itself defined in terms of an integral. These circumstances prevent the use of standard methods like inverse transform sampling.

Note that there is an implicit assumption that C remains finite, which requires that the transmittance decays sufficiently quickly. In practice, the scene would likely contain only media within limited spatial extent, in which case sampling can be restricted to this volume to avoid this technicality. We have accounted for this assumption by setting the integral's upper limit to t_s, the location of the closest surface or medium bounding box along the ray.



(a) Medium properties



(b) Delta Tracking (c) Ratio Tracking (d) Residual Ratio Tracking

Figure 5.6: Demonstrating our novel transmittance sampling methods based on delta- and (residual) ratio tracking. **(a**, bottom**)** We use a 1D analytic medium with density following a sinusoidal function. The chosen majorant and control densities are also shown. **(a**, top**)** Forward rendering of such a medium involves sampling free-flight distances proportionally to $\sigma_t(t)\,\mathrm{T}(t)$. Locations where $\sigma_t \approx 0$ receive few or no samples, which leads to bias and high variance when using this same sampling technique to estimate gradients. Instead, we propose dedicated transmittance sampling techniques for the adjoint. Delta tracking **(b)**, ratio tracking **(c)** and residual ratio tracking **(d)** can be interpreted as building estimates of the transmittance function using binary, piecewise constant and piecewise-exponential functions respectively. We build transmittance sampling algorithms from those trackers by sampling distances from their individual function approximations. They produce weighted samples (green histogram) with density equal to the transmittance function T, where the weights (yellow histogram) approximate the normalization constant of T.

It would seem that the presence of the normalizing constant C forces us to revise the estimator of Equation (5.11) to

$$\langle \partial L_{1a}^{\mathrm{DRT}} \rangle = C\, L_s(t')\; \partial_{\boldsymbol{\theta}} \big[\sigma_t(t')\, \alpha(t') \big] \,. \tag{5.14}$$

However, instead of evaluating Equation (5.14) using unweighted samples that are ideally proportionally to the re-normalized transmittance, we pursue a strategy that generates *weighted* samples. In other words, the samples alone will not have the correct distribution, but they will be distributed according to p_{T} when considered along with their weights. These weights will also serve a secondary purpose: to provide an unbiased estimate of the normalization constant C. Serendipitously, all of this is possible with a simple extension to the family of delta- and ratio trackers.

Delta tracking. We start with a simple strategy based on the delta tracking free-flight distance sampling method. On a given ray segment $[0, t_s)$, delta tracking repeatedly samples steps t_0 from a homogenized medium with density $\bar{\sigma} \geq \sigma_t(t)\ \forall t$. At each step, the nature of the interaction is determined stochastically: a *real* particle is encountered with probability $\sigma_t(t)/\bar{\sigma}$. Otherwise, a *null* particle is encountered and the traversal continues.

If a real interaction occurs at $t_1 \in [0, t_s)$, light is scattered or absorbed and the ray throughput falls to zero. The free-flight distance t_1 is sampled with probability

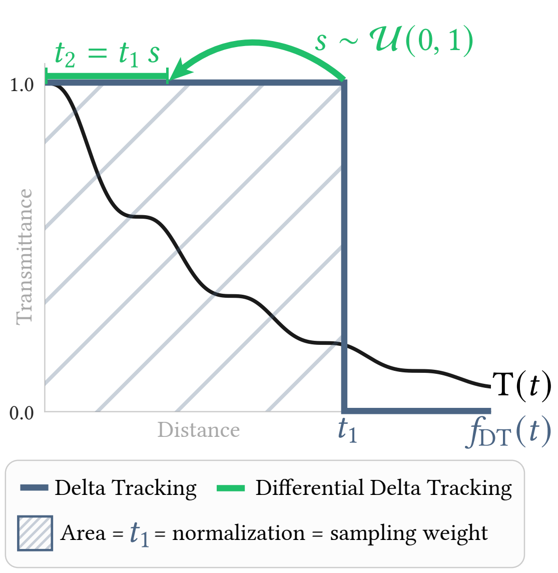
$$p_1(t) = \sigma_t(t)\ \mathrm{T}(t).$$

We can interpret this randomized process as creating the following binary function or "tracking":

$$f_{\mathrm{DT}}(t) = \begin{cases} 1 & \text{if } 0 < t \leq \min(t_1, t_s), \\ 0 & \text{otherwise.} \end{cases}$$

An example tracking is drawn in Figure 5.7 (left, blue curve). By averaging together many such trackings, we obtain an unbiased estimate of transmittance at all points. In Figure 5.6, we draw individual trackings as thin red lines, and the averaged function estimate in blue.

Sampling from a medium realization. We could interpret each sampled tracking f_{DT} as the *actual* transmittance function of a particular realization of the stochastic medium. We propose sampling proportionally to that transmittance function and show



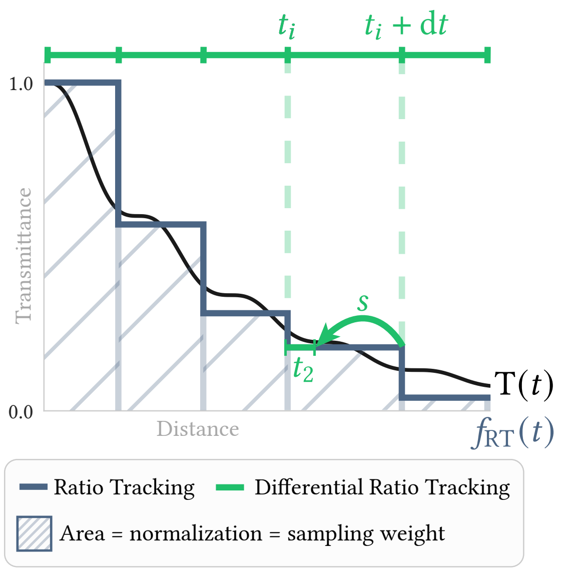


Figure 5.7: Our proposed sampling methods build on the family of delta- (left) and ratio (right) trackers. The trackers build piecewise approximations f of the true transmittance function T. We simply sample a distance proportionally to the area under f which, together with a sampling weight equal to the total area under the curve, yields unbiased samples of T.

that the resulting density exactly corresponds to our target. Given $t_1 \sim p_1$ sampled by delta tracking, we simply sample a new distance

$$t_2 \coloneqq t_1\, s, \quad \text{with } s \sim p_s(s) \coloneqq \mathbb{1}_{[0,1)}(s).$$

This step is illustrated in Figure 5.7 (left, green arrow). Let us examine the corresponding product density

$$\begin{aligned}\tilde{p}_2(t) &= \int_{-\infty}^{+\infty} p_1\left(\frac{t}{s}\right) p_s(s)\, \frac{1}{|s|}\, \mathrm{d}s \\ &= \int_0^1 p_1\left(\frac{t}{s}\right) \frac{1}{s}\, \mathrm{d}s \\ &\overset{(1)}{=} \int_t^{+\infty} p_1(q)\, \frac{1}{q}\, \mathrm{d}q\,,\end{aligned}$$

where step (1) involves the change of integration variable $q \coloneqq {}^{t}/_{s}$. Unfortunately, $\tilde{p}_2$ does not quite correspond to our desired density. We correct it by associating a weight

$w := q = {}^{t}\!/_{s}$ to each sample t. The weighted density is

$$\begin{aligned} p_2(t) &= \int_{-\infty}^{+\infty} p_1\left(\frac{t}{s}\right) p_s(s)\,\frac{t}{s}\,\frac{1}{|s|}\,\mathrm{d}s \\ &\overset{(1)}{=} \int_{t}^{+\infty} p_1(q)\,\mathrm{d}q \\ &= \int_{t}^{+\infty} \sigma_t(q)\,\mathrm{T}(0,q)\,\mathrm{d}q \\ &= \mathrm{T}(0,t) \int_{t}^{+\infty} \sigma_t(q)\,\mathrm{T}(t,q)\,\mathrm{d}q\,, \end{aligned} \tag{5.15}$$

where (1) denotes the same change of variable as before. Georgiev et al. [2019, Equation (9)] have shown that transmittance can be expressed as the following Volterra integral equation:

$$\mathrm{T}(a,b) = 1 - \int_{a}^{b} \sigma_t(s)\,\mathrm{T}(s,b)\,\mathrm{d}s\,.$$

Substituting into Equation (5.15) yields

$$\begin{aligned} p_2(t) &= \mathrm{T}(0,t)\ \left(1 - \mathrm{T}(t,+\infty)\right) \\ &= \mathrm{T}(0,t) - \mathrm{T}(0,+\infty)\,. \end{aligned}$$

Finally, using our earlier assumption that the normalization constant C is finite[3], we have $\mathrm{T}(0,+\infty) = 0$ and obtain

$$p_2(t) = \mathrm{T}(0,t)\,.$$

We conclude that the weighted samples $(t_2, w = {}^{t_2}\!/_{s})$ have the desired distribution and call this sampling technique *differential delta tracking*. In Figure 5.6b, we confirm experimentally that the weighted density matches the transmittance of an example 1D medium slice.

Normalization constant. Further inspection reveals that the chosen sampling weight is quite meaningful: ${}^{t_2}\!/_{s} = t_1$ is exactly the area under the curve of the binary function f_{DT}. In other words, it captures the difference between our unnormalized target density T and the normalized density we are actually sampling from. Moreover, by construction of delta tracking, the trackings approximate the transmittance function [148]:

$$\mathbb{E}\left[\int_{0}^{t_s} f_{\mathrm{DT}}(s)\,\mathrm{d}s\right] = \mathbb{E}[t_1] = \int_{0}^{t_s} \mathrm{T}(s)\,\mathrm{d}s = C\,. \tag{5.16}$$

[3]In practice, for volumes defined within finite spatial extents, this is equivalent to setting $\sigma_t = +\infty$ beyond the volume's domain.

The sampling weight w therefore takes on the role of C in Equation (5.14), completing the estimator.

5.4.3 Differential ratio tracking

Using the same intuition, we can build a more efficient transmittance sampling technique based on *ratio tracking* [148], which is a simple and effective strategy that expands on delta tracking to compute unbiased estimates of the transmittance along a ray segment $(0, t_s)$. As before, successive steps t_i are sampled in closed form from a homogenized medium with density $\bar{\sigma} \geq \sigma_t(x)\ \forall x$. However, rather than sampling a binary real/null decision at each step, the probability of interacting with a real particle is used to update the estimated transmittance up to the current point. This can be interpreted as building a piecewise constant, rather than binary, tracking. It approximates the transmittance function more closely:

$$f_{\mathrm{RT}}(t) = \prod_{t_i \leq \min(t, t_s)} 1 - \frac{\sigma_t(t_i)}{\bar{\sigma}},$$

where $\sigma_t(t_i)/\bar{\sigma}$ is the probability of interacting with a real particle. An example piecewise constant tracking f_{RT} is drawn in Figure 5.7 (right, blue curve).

Sampling from ratio trackings. Following the same intuition as in Section 5.4.2, we derive a transmittance sampling technique from the ratio tracking algorithm. Where differential *delta* tracking simply needed to sample from the binary function f_{DT}, we now need to sample from the piecewise constant f_{RT}.

Given a realization f_{RT}, it is easy to build and sample from a discrete 1D distribution based on the area under each constant segment. Once a segment has been selected, we simply sample uniformly along it. This step is illustrated in Figure 5.7 (right, green arrow). As before, we associate a sampling weight w equal to the area under the curve of f_{RT}:

$$w = \sum_{t_i} (t_{i+1} - t_i)\, f_{\mathrm{RT}}(t_i)\,, \tag{5.17}$$

$$\text{with} \quad \mathbb{E}[w] = \mathbb{E}\left[\int_0^{t_s} f_{\mathrm{RT}}(s)\,\mathrm{d}s\right] = \int_0^{t_s} \mathrm{T}(s)\,\mathrm{d}s = C\,. \tag{5.18}$$

Due to the individual trackings' closer match to the transmittance curve, the sampling weights also converge faster to C.

Online segment sampling. The sampling method described above involves fully constructing a tracking f_{RT} before sampling proportionally to its area. While this is certainly feasible, the storage and computational overhead would be unnecessarily high, especially considering that the number of segments is not known ahead of time and grows with the majorant $\bar{\sigma}$ and total distance t_s.

Instead, we turn to *weighted reservoir sampling* [153] in order to sample a segment *online*, simultaneously with ratio tracking. Desirably, reservoir sampling uses $\mathcal{O}(1)$ memory and has very limited computational overhead. Its pseudocode is included in Listing 4.

```
class Reservoir:
    y = 0
    w_sum = 0
    n = 0
    def update(x, w):
        w_sum += w
        n += 1
        if rand() < w / w_sum:
            y = x

def reservoir_sampling(samples):
    r = Reservoir()
    for x, w in samples:
        r.update(x, w)
    return r.y
```

Listing 4: Pseudocode for the weighted reservoir sampling algorithm of Chao [153]. Importantly, the reservoir update can be carried out *online*, as samples are being produced and without storing them. We will use this fact in Listing 5. Adapted from [154, Algorithm 2].

At each step $t_i \longrightarrow t_{i+1}$ of ratio tracking, we propose to the reservoir sampling algorithm the corresponding segment of f_{RT} with a weight equal to the area under it:

$$w_i = d_i \, f_{\mathrm{RT}}(t_i) \,, \quad \text{with } d_i = t_{i+1} - t_i \,.$$

The reservoir is updated to this segment with probability $w_i / \sum_{j=1}^{i} w_j$. Once ratio tracking terminates by reaching the maximum distance t_s, we read the values (t_r, d_r) from the reservoir. By construction of reservoir sampling, this segment was sampled with probability $d_r f_{\mathrm{RT}}(t_r) / \sum_j d_j \, f_{\mathrm{RT}}(t_j)$. Moreover, the overall sample weight $w = \sum_j d_j \, \mathrm{T}(t_j)$ is readily available as it is computed as part of the reservoir sampling algorithm.

We will refer to this estimator as *differential ratio tracking*. Its full pseudocode is given in Listing 5. We also demonstrate it in action on a 1D example in Figure 5.6c.

```
def drt_sample($t_{\max}$):
    Tr = 1.
    $t_{\mathrm{run}}$ = $w_{\mathrm{acc}}$ = 0.

    while $t_{\mathrm{run}}$ < $t_{\max}$ :
        # Sample from the homogenized medium
        dt = $-\log(1-\mathrm{rng}())/\bar{\sigma}$
        dt = min(dt, $t_{\max}$ - $t_{\mathrm{run}}$ )
        # Propose current constant segment with weight
        # equal to area under the curve.
        $w_{\mathrm{step}}$ = Tr * dt
        $w_{\mathrm{acc}}$ += $w_{\mathrm{step}}$
        if (rng() * $w_{\mathrm{acc}}$ ) < $w_{\mathrm{step}}$ :
            # Update reservoir (happens at least once)
            reservoir = ($t_{\mathrm{run}}$ , dt)

        # Update transmittance estimate
        Tr *= 1 - ($\sigma_t(t_{\mathrm{run}})$ / $\bar{\sigma}$)
        $t_{\mathrm{run}}$ = $t_{\mathrm{run}}$ + dt

    # Final uniform sampling over the chosen segment
    t, dt = reservoir
    t = t + rng() * dt
    assert t ≤ $t_{\max}$
    return t, $w_{\mathrm{acc}}$
```

Listing 5: Differential ratio tracking efficiently samples t proportionally to transmittance $\mathrm{T}(t)$ along the current ray up to $t_{\max}$ by combining ratio tracking and reservoir sampling. The sampling weight is an unbiased estimate of the integral of transmittance $C = \int_0^{t_s} T(s)\,\mathrm{d}s$.

Differential residual ratio tracking. Finally, we build a third sampling technique based on *residual ratio tracking* [148]. Residual ratio tracking can be seen as applying ratio tracking to a "residual medium" whose density is equal to the original density minus a *control density* σ_c. The trackings are therefore further improved from piecewise constant to piecewise-exponential. Some example trackings are shown in Figure 5.6d.

In practice, we found that the limited improvement in gradient variance did not warrant the inclusion of an additional hyperparameter (the control density). Our experiments were therefore conducted with differential *ratio* tracking. For completeness, the pseudocode is nevertheless given in Listing 6.

```
def drrt_sample($t_{\max}$):
    Tr = 1.
    $t_{\text{run}}$ = $w_{\text{acc}}$ = 0.

    while $t_{\text{run}}$ < $t_{\max}$ :
        # Sample from the homogenized medium
        dt = $-\log(1-\text{rng}())/\bar{\sigma}$
        dt = min(dt, $t_{\max}$ - $t_{\text{run}}$ )

        # Propose current exponential segment with weight
        # equal to area under the curve.
        $w_{\text{step}}$ = Tr * (1 - exp(-dt $\sigma_c$)) / $\sigma_c$
        $w_{\text{acc}}$ += $w_{\text{step}}$
        if (rng() * $w_{\text{acc}}$ ) < $w_{\text{step}}$ :
            # Update reservoir (happens at least once)
            reservoir = ($t_{\text{run}}$ , dt)

        # Update transmittance estimate
        Tr *= exp(-$\sigma_c$ dt) * (1 - ($\sigma_t(t_{\text{run}}) - \sigma_c$) / $\bar{\sigma}$)
        $t_{\text{run}}$ = $t_{\text{run}}$ + dt

    # Final sampling over the chosen exponential segment
    t, dt = reservoir
    t -= log(1 - rng() * (1 - exp(-$\sigma_c$ dt))) / $\sigma_c$
    assert t <= $t_{\max}$
    return t, $w_{\text{acc}}$
```

Listing 6: Differential residual ratio tracking pseudocode. σ_c is the control density and $\bar{\sigma}$ the majorant of the residual medium: $\bar{\sigma} \geq |\sigma_t - \sigma_c|$. Similarly to differential *ratio* tracking (Listing 5), we combine the residual ratio tracking transmittance estimator with reservoir sampling to produce weighted samples t with density equal to the transmittance function.

5.4.4 Multiple importance sampling

We now have at our disposal the standard free-flight technique, sampling proportionally to $\sigma_t \mathrm{T}$ as well as our novel technique, sampling proportionally to T only. As was mentioned in Section 5.4.1, the former is well suited to sample all but the in-scattering gradient term.

Our sampling technique is used to estimate in-scattering gradients, with estimator $\langle \partial L_{1a}^{\mathrm{DRT}} \rangle$. Since we will construct paths using free-flight sampling in any case (and the incident radiance term L_i at that point is provided by path replay), we may as well combine the corresponding in-scattering gradients $\langle \partial L_1^{\mathrm{DT}} \rangle$ with ours using multiple importance sampling [19]. Luckily, the multiple importance sampling weights themselves are simple to compute. For example, using the power heuristic:

$$\begin{aligned} p_{\mathrm{DT}}(t) &= \sigma_t(t)\, T(t)\,, \quad p_{\mathrm{DRT}}(t) = T(t)\,, \\ w_{\mathrm{DT}}(t) &= \frac{p_{\mathrm{DT}}(t)^2}{p_{\mathrm{DT}}(t)^2 + p_{\mathrm{DRT}}(t)^2} = \frac{\sigma_t(t)^2}{\sigma_t(t)^2 + 1}\,, \qquad (5.19) \\ \text{and} \quad w_{\mathrm{DRT}}(t) &= \frac{1}{\sigma_t(t)^2 + 1}\,. \qquad (5.20) \end{aligned}$$

Using multiple importance sampling, we improve variance at negligible extra cost[4].

5.4.5 Preserving linear time complexity

Path replay backpropagation (Section 5.2.2) can estimate derivatives in linear time. It is desirable that our method preserves this property. Unfortunately, if Equation (5.11) were used as stated, our estimator would require an additional estimate of incident illumination $L_i(t')$. This is because the distance t' sampled by our estimator differs from the distance sampled to build the path; thus, path replay does not provide the needed value $L_i(t')$. Estimating $L_i(t')$ under global illumination requires tracing a recursive path, resulting in an overall $O(n^2)$ cost when repeated at every bounce of the original path.

Instead, to maintain $O(n)$ cost, we select only *a single bounce* of the original path, proportionally to the throughput $\mathrm{Tr}(d)$ of the corresponding path segment, at which we evaluate $\langle \partial L_{1a}^{\mathrm{DRT}} \rangle$. To this end, we again use weighted reservoir sampling as the path is traversed. The reservoir is updated with a probability proportionally to $\mathrm{Tr}(d)$ and records the segment index as well as the necessary state to spawn the recursive path.

[4]Since $p_{\mathrm{DRT}}(t)$ is unnormalized, variance reduction is neither guaranteed nor governed by the bounds proven by Veach and Guibas [19]. Nonetheless, we observe significant variance reduction in practice; see Figure 5.10d.

Due to spawning only a single recursive path, its resulting estimate must be weighted by $w = \sum_{d=0}^{n} \mathrm{Tr}(d)$ (the unnormalized inverse probability of being selected), where n is the path length. In Figure 5.8, we empirically validate that—while this optimization increases gradient variance—it remains well below that of the free-flight based estimator.

5.5 Evaluation

We now turn to the evaluation of our proposed gradient estimator, validating its correctness and efficiency. Our method will be used again in Chapter 10, where we will consider an inverse volume rendering application, and how to avoid the inherent suboptimal local minima.

5.5.1 Correctness and variance

Validation against finite differences. We have established that gradients computed using free-flight sampling-based estimators can suffer from bias and high variance. Figure 5.8 provides empirical verification: we compare the mean gradients and their standard deviation computed by several methods on the same dense medium, lit by a realistic high dynamic range environment map. Specifically, we compare reference gradients, computed by brute force using finite differences (FD)—hence the low resolution—with free-flight sampling, defensive sampling (Equation (5.9)), and two configurations of our method: an $O(n^2)$ implementation where our estimator is used at every path segment, as well as our linear time variant.

As expected, gradients estimated using free-flight sampling exhibit bias in empty regions ($\sigma_t = 0$, striped area), as well as high standard deviation, manifesting as large outliers, where density is low. Using defensive sampling helps eliminate the largest outliers, but standard deviation remains high due to the mismatch between the sampling density and integrand. Our estimator yields correct gradients as well as significantly lower standard deviation in both the $O(n^2)$ and $O(n)$ configurations.

Validation against analytic gradients. We further validate our method's correctness by comparing the computed gradient to analytic ground truth values in a simplified case. We consider a homogeneous, fully forward scattering, non-emissive, single-scattering medium. The volume rendering equation (5.1) then simplifies to:

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = \int_0^{t_s} \mathrm{T}(0, t)\, \sigma_s(t)\, L_s(t, \boldsymbol{\omega})\, \mathrm{d}t + \mathrm{T}(0, t_s)\, L_e(t_s). \tag{5.21}$$

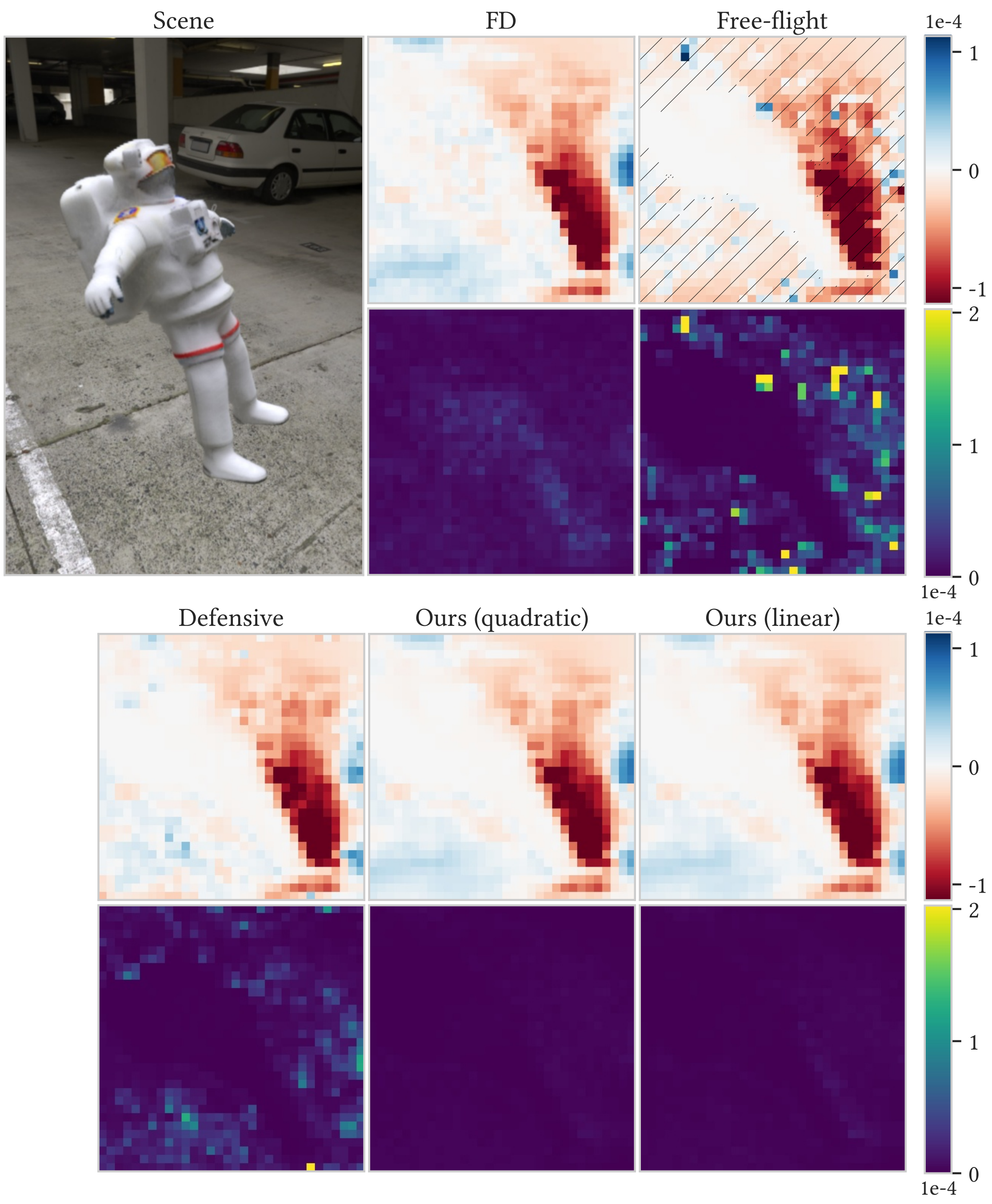


Figure 5.8: We validate gradients with respect to medium density σ_t in a synthetic scene (top left). The mean gradient values (blue-red visualization) and standard deviation (purple-yellow visualization) are shown for the slice $z = 15$ of the dense σ_t 3D parameter grid. Finite differences (FD) provide reference gradient values, but cannot be used in the inner loop of an optimization due to the prohibitive runtime. Using a free-flight sampling based estimator (Section 5.3.2) to estimate gradients results in bias where $\sigma_t = 0$ (striped area) and high variance where $\sigma_t \approx 0$ (also shown in Figure 5.2). Defensive sampling (Section 5.3.2) with $\epsilon = 0.1$ mitigates the largest outliers, but does not eliminate the remaining variance. Our novel sampling technique, whether used at each path segment ("quadratic", Section 5.4.1) or once per path ("linear", Section 5.4.5) addresses both the bias and variance concerns.

Since the medium is fully forward scattering, $f_s(x, \omega, \boldsymbol{\omega}') = \delta(\boldsymbol{\omega} - \boldsymbol{\omega}')$ and therefore $L_s(t, \boldsymbol{\omega}) = L_i(t, \boldsymbol{\omega})$:

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = \sigma_t \, \alpha \int_0^{t_s} \mathrm{T}(0, t) \, L_i(t, \boldsymbol{\omega}) \, \mathrm{d}t + \mathrm{T}(0, t_s) \, L_e.$$

After an interaction within the single-scattering medium at t, the ray continues without further scattering, therefore $L_i(t, \boldsymbol{\omega}) = \mathrm{T}(t, t_s) \, L_e(t_s)$. This drastically simplifies the expression of incident radiance:

$$\begin{aligned} &= \sigma_t \, \alpha \int_0^{t_s} \mathrm{T}(0, t) \, \mathrm{T}(t, t_s) \, L_e \, \mathrm{d}t + \mathrm{T}(0, t_s) \, L_e \\ &= \sigma_t \, \alpha \, L_e \int_0^{t_s} \mathrm{T}(0, t_s) \, \mathrm{d}t + \mathrm{T}(0, t_s) \, L_e \\ L_i(\mathbf{x}, \boldsymbol{\omega}) &= L_e \, \mathrm{T}(t_s) \; (\sigma_t \, \alpha \, t_s + 1) \,, \end{aligned} \tag{5.22}$$

which is easily evaluated in closed form.

We create a scene containing such a single-scattering medium with $t_s = L_e = 1$. We then estimate density and albedo gradients using both the free-flight based estimator (Section 5.3.2) and ours (Section 5.4.1). The results are plotted as a function of the medium's density value in Figure 5.9. While all methods match the analytic gradients for all $\sigma_t > 0$, density gradients estimated with the free-flight estimator suffer from high variance as σ_t approaches zero. Our estimator computes unbiased and low-variance gradients.

Ablation study. We study the contribution of each component of our method in Figure 5.10. The test medium contains empty, thin, and dense regions, and is lit by a realistic outdoor environment map. Given initial medium density ($\sigma_t(\mathbf{x})$, shown at the top left) and albedo parameters, we estimate gradients with respect to $\sigma_t(\mathbf{x})$ using different estimators. Density is represented by a trilinearly interpolated dense grid with resolution $256 \times 128 \times 128$ (more than four million parameters). We then visualize the standard deviation of gradients w.r.t. each σ_t value in a 2D slice of that grid.

We observe a very large standard deviation in regions of low density when estimating gradients with the free-flight sampling-based estimator **(a)**. Note that due to in-scattering gradients being entirely missing in empty regions ($\sigma_t = 0$, striped area denoting bias), variance is artificially low. Our method **(b)** fully eliminates gradient outliers. In regions where $\sigma_t = 0$, variance is moderately increased due to the (correct) inclusion of the in-scattering gradient term. Reducing the sampling frequency from every path

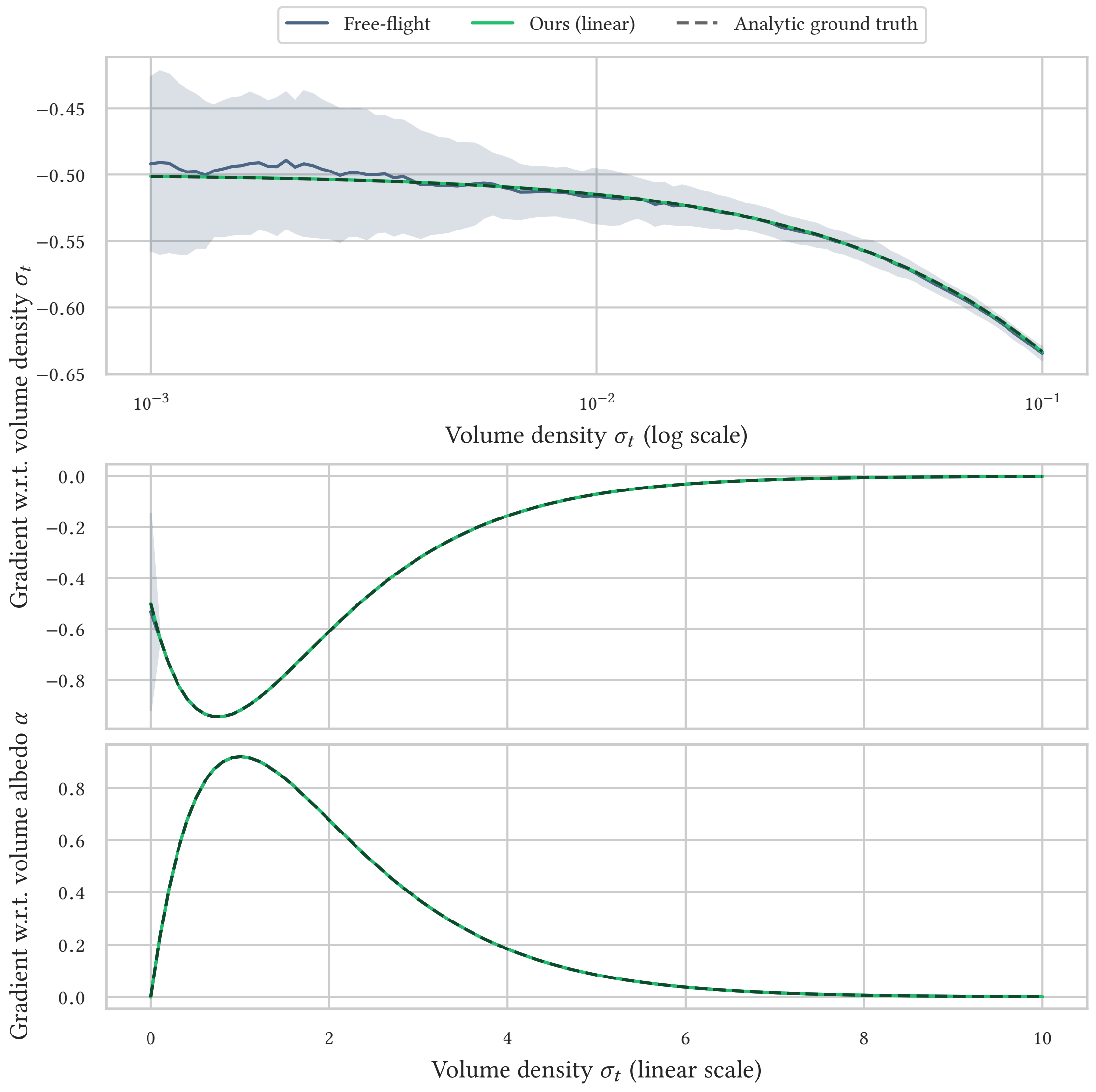


Figure 5.9: In a single-scattering, fully forward scattering homogeneous medium, we compare the gradients computed by the standard free-flight based estimator and our method to the ground truth analytic gradients. While all methods compute correct gradients in this setting for all $\sigma_t > 0$, we note once again that free-flight based gradients suffer from high standard deviation as σ_t approaches zero (shaded area, topmost plot). With our estimator, the standard deviation is low enough not to appear on the plots. Albedo gradients are estimated correctly and efficiently by all methods (third plot).

segment to a single segment **(c)** brings our estimators' complexity back to linear time (Section 5.4.5), but also increases variance in regions of low throughput such as the center of the dense smoke plume. However, this is corrected by combining our estimator with regular free-flight sampling via multiple importance sampling **(d)** (Section 5.4.4). We find that switching to an estimator based on *residual* ratio tracking **(e)** rather than ratio tracking does not bring noticeable improvements. Likewise, increasing the medium's majorant from 1.01× to 10× the largest σ_t value does not significantly improve gradient variance **(f)**. In order to avoid the additional hyperparameter (medium control density) and additional computation cost respectively, we recommend using differential *ratio* tracking and a majorant of $\bar{\sigma} = 1.01\ \max_{\mathbf{x}} \sigma_t(\mathbf{x})$.

5.5.2 Role of the optimizer

When optimizing over a large parameter space with noisy gradients, the choice of optimization algorithm (optimizer) has a significant impact on convergence. As we have seen, gradients computed with free-flight sampling-based estimators are prone to large outliers (Figure 5.10a). Using stochastic gradient descent, either with (SGDm) or without momentum (SGD), these gradient outliers are almost directly reflected in the descent steps. In practice, regions of the medium that should be fully empty end up being filled with artifacts; see Figures 5.1, 10.4 and 10.6. The outliers occur frequently enough that the optimization can never recover. In contrast, variance-adaptive optimizers such as Adam [150] are effective at suppressing such outliers—to a perhaps surprising degree. The outliers merely trigger correspondingly low step sizes. Because the outliers only occur in regions where $\sigma_t \approx 0$, the *per-parameter* step size being small in those regions of the volume's density grid does not prevent the rest of the medium parameters from converging[5]. Altogether, optimizations converge to comparatively good solutions when using Adam, despite the numerous outliers.

5.5.3 Implementation and performance

Compared with pure free-flight sampling in the path replay backpropagation framework [21], our $O(n)$ algorithm requires tracing one additional recursive path for each training path. Table 5.1 shows that the recursive path amounts to an overhead of ~20%

[5]Note this is only possible because the grid-based parametrization used in our experiments maps parameter space directly to world space. If the medium was defined as *e.g.* the result of a procedural computation, the aggressive reduction of step sizes may slow or prevent overall convergence. Our method's reduced gradient variance is then even more advantageous.

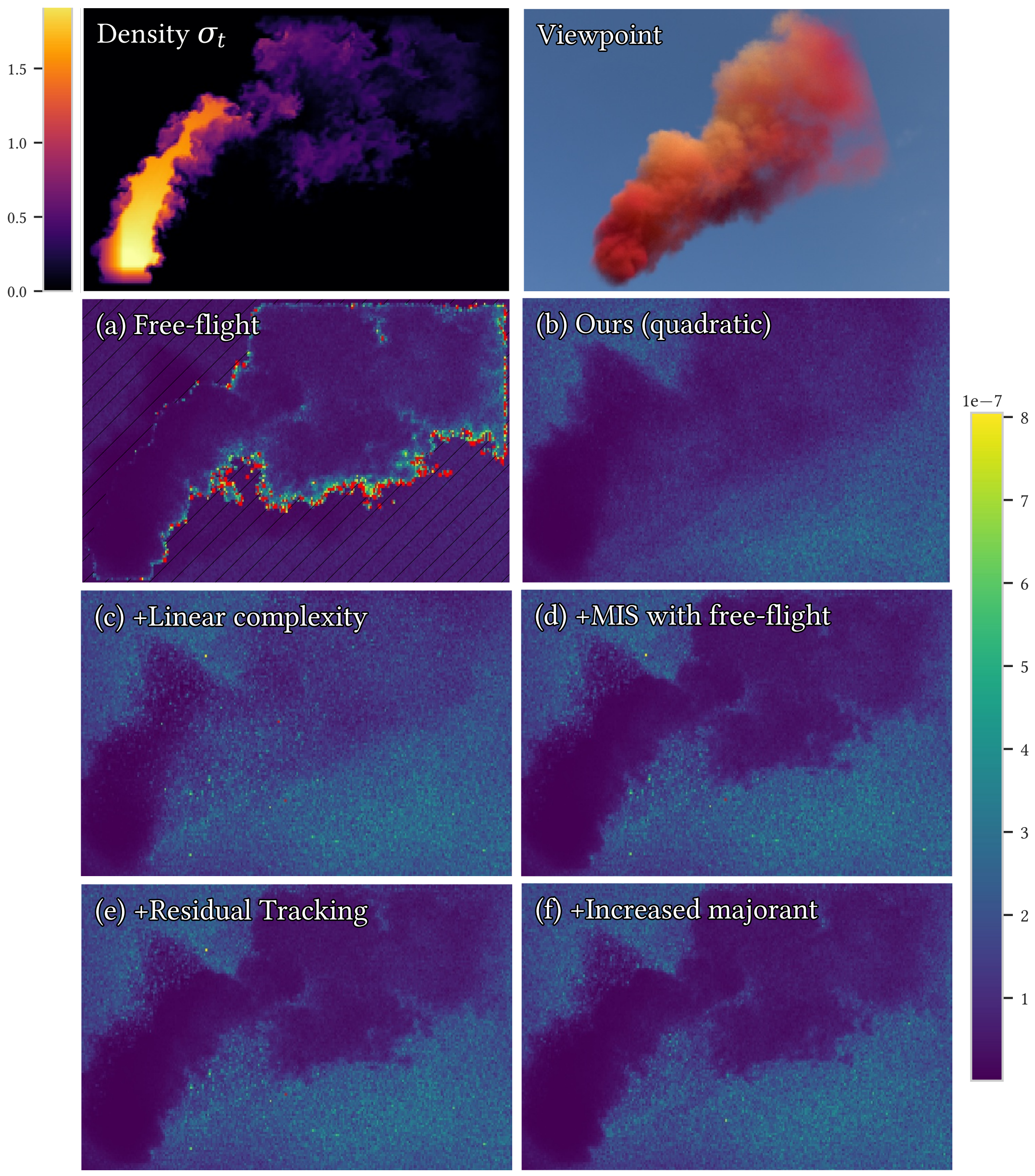


Figure 5.10: Ablation study. We compare the standard deviation of gradients with respect to the density σ_t of the medium shown in the top-left rendering. Inverse rendering of this medium is shown in Figure 10.6). The medium includes regions of high, low and zero density (bottom left). We show slice $z = 64$ of parameter space. **(a)** As was seen previously (Figure 5.8), free-flight based gradients are prone to large outliers where $\sigma_t \approx 0$. Values outside of the visualization range are denoted in red. In regions where $\sigma_t = 0$ (striped area), the missing in-scattering gradients (bias) lead to artificially low standard deviation. **(b)** Our unbiased *differential ratio tracking* estimator resolves both the bias and variance issues when used on every path segment (leading to quadratic complexity). **(c)** Selectively using our estimator once per path lowers the complexity back to linear time, but introduces additional variance in denser regions where path throughput is lower. However, combining both techniques with multiple importance sampling **(d)** makes up for it at virtually no cost. Finally, switching to differential *residual* ratio tracking **(e)** or increasing the medium majorant tenfold **(f)** does not seem to significantly improve standard deviation.

Table 5.1: We report the median cost per training iteration (s/it) for vanilla path replay backpropagation [21] (free-flight) as well as with our proposed $O(n)$ sampling technique enabled. Our extension involves tracing an additional recursive path, which amounts to a ∼20% overhead. To ensure the same experimental conditions, both methods compute gradients w.r.t. the same starting medium, of which we report the mean optical depth and maximum density. In practice, different optimization techniques lead to different media, which additionally affects performance.

	Optical depth	Density σ_t	Median runtime (s/it)	
	mean	max	Free-flight	Ours
Dust devil	2.12	45.99	0.20	0.24
Red cow	2.45	12.87	0.32	0.37
Smoke plume	3.24	95.57	0.24	0.29

in our implementation, provided that both algorithms compute gradients w.r.t. the same starting medium[6]. We report the median runtime per iteration on three different scenes with different densities. One iteration includes the three steps of path replay backpropagation described in Section 5.2.2, with next event estimation and a maximum path length of 64. The timings were measured on an NVIDIA RTX 3090 GPU over 50 runs.

We implemented both methods in Mitsuba 3 [79]. Our implementation runs efficiently in reverse-mode and on the GPU, making it possible to optimize many parameters in parallel: the medium density σ_t is typically represented by a 256×256×256 dense grid, and the single-scattering albedo α by a 256×256×256×3 grid for a total of more than *67 million* parameters. It is therefore possible to optimize scattering volumes with full unbiased global illumination, perhaps contrary to common belief.

Beyond important features such as next-event estimation, our implementation includes support for a spatially-varying majorant (to which our method is agnostic). Allowing the majorant to vary on a coarse grid becomes important for performance when the medium's bounding box contains both very dense and very thin or empty regions.

Note that there remains room for optimization, for example using an adaptive voxel grid to store medium properties, porting performance-critical functions to CUDA or adaptively sampling primary rays based on the re-rendering error.

[6]In practice, different optimization techniques lead to different media, which additionally affects performance as the number of path vertices increases with the density of the medium.

5.6 Conclusion

Among all methods for physically based differentiable rendering, volumetric representations have been particularly successful, due in part to their seemingly trivial differentiability. Our work shows that severe issues persist in existing gradient estimation methods, leading to biased and high-variance gradients. Adaptive optimizers such as Adam reduce the impact of poor-quality gradients, which may be the reason these issues were not noticed at first. We present a simple, unbiased method tailored to the estimation of gradients with respect to the parameters of scattering volumes. Our estimator retains the linear time and constant memory complexity of the state-of-the-art differentiable rendering algorithm [21], allowing the efficient estimation of gradients with respect to millions of parameters in parallel.

We believe the development of new sampling techniques dedicated to gradients estimation is a highly promising direction for future work.

The method proposed in this chapter will be put into practice in Chapter 10, with a challenging inverse volume rendering application. We will see how to leverage a non-physical emissive volume model to bootstrap the optimization of scattering volumes, thus avoiding suboptimal local minima.

Part II

Systems

6 | Systems for physically based differentiable rendering

Relevant background: Section 2.3.

After establishing three methods to compute parameter gradients in reverse mode, we turn to the design of systems allowing effective implementation and execution of those algorithms.

6.1 Scale of rendering systems

Because they simulate the intricacies of the visual world, physically based rendering systems tend to be large and complex: for instance, PBRT v3 [24] and Mitsuba 0.6 [26] consist of over 60 and 180 thousand lines of C++ code, respectively. Industrial rendering systems are larger still, with typical sizes on the order of one million lines of code.

Adaptability. Despite their size, these systems lack many features of growing importance: for instance, predictive rendering applications require that simulations correctly account for the effects of both spectral transport and polarization. In theory, this is a straightforward extension: one must simply replace RGB radiance and reflectance values with wavelength-dependent Stokes vectors and Mueller matrices and update a few models that are directly affected by these phenomena, *e.g.* by switching from a spectrally constant refractive index to Cauchy's equation and adopting the complex-valued form of the Fresnel equations. In practice, the modification changes the representation of quantities central to any renderer, requiring a substantial redesign of the entire system.

Vectorized rendering. Vectorized rendering systems, such as MoonRay [155], Iray [156] and Hyperion [157], leverage Single Instruction/Multiple Data (SIMD) units on modern CPUs and GPUs to efficiently sample many light paths in parallel, thereby reducing the overall computation time. Efficient vectorization generally involves a mechanical translation into a sequence of *compiler intrinsics* or specialized compiler infrastructure to generate SIMD machine instructions, algorithms with improved coherence[1], and data structures arranged in *structure of arrays* (SoA) form. The latter requires transposing the

[1] *i.e.* regular control flow and memory accesses with spatio-temporal locality.

memory layout of the entire application, which constitutes another example of a highly intrusive change to every system component. Designing vectorized renderers remains a time-consuming endeavor—for instance, the MoonRay project marks the result of a concerted four-year development effort of a team of engineers [155, 158].

6.2 Differentiating through renderers

Of particular interest to us are the challenges associated with the computation of derivatives through complex physically based rendering algorithms.

Manual derivation. Li et al. [57] presented the first comprehensive technique for differentiable rendering that accounts for all salient transport effects including discontinuities. Its freely available implementation, Redner, reveals the inherent challenges of realizing such systems: discontinuities aside, the basic loop of the underlying path tracer (which would likely be realizable using at most 200 lines of C++ code in a "classical" renderer) expands into approximately 3000 lines of hand-written derivative code partitioned over multiple CUDA kernels that communicate through large groups of auxiliary buffers. Adding a new model to the system entails manual differentiation of all relevant expressions with careful consideration of what information must be cached when and where, so that it can be passed from kernel to kernel during subsequent gradient propagation passes. More advanced (but well-understood) bidirectional or volumetric techniques present a formidable challenge in this context.

Existing AD frameworks. The systems presented in this thesis are related to tools like TensorFlow [61] and PyTorch [159] but address fundamental problems that arise in the context of rendering. Both PyTorch and TensorFlow provide two main operational modes: *eager mode* directly evaluates arithmetic operations on the GPU, which yields excellent performance in conjunction with arithmetically intensive operations like convolutions and large matrix-vector multiplications, both of which are building blocks of neural networks. When evaluating rendering code created from much simpler arithmetic, the resulting memory traffic and scheduling overheads induce severe bottlenecks.

The second operational mode requires an up-front specification of the complete computation graph to generate a single optimized GPU kernel (*e.g.* via XLA in TensorFlow and `jit.trace` in PyTorch). This is feasible for neural networks, whose graph specification is very regular and typically only consists of a few hundred operations. Rendering code, on the other hand, involves much larger graphs, whose structure is *unpredictable*:

program execution could jump to almost any part of the system when rendering a complex scene. The full computation graph would simply be the entire codebase (~100K lines of code), which is of course far too big.

The Enoki library of Jakob [13], presented in Section 7.2.3, was designed to handle the intricacies of physically based rendering and could be interpreted as a middle ground between the two extremes discussed above. With Enoki's lazy just-in-time (JIT) compiler, graphs are created on the fly while simulating the process of scattering and transport and tend to be several orders of magnitude larger compared to typical neural networks. They consist mostly of unstructured and comparably simple arithmetic and are lazily fused into optimized CUDA kernels. Since our system works without an upfront specification of the full computation graph, it must support dynamic indirection via virtual function calls that can simultaneously branch to multiple different implementations. The next chapter introduces Mitsuba 2, our retargetable primal and adjoint renderer based on Enoki.

7 | Mitsuba 2

Relevant background: Sections 2.3, 2.5 and 2.6.

Mitsuba 2 is the result of a multi-year development effort from a number of contributors. Notably, Prof. Wenzel Jakob was the project lead, and developed the Enoki library and many aspects of Mitsuba 2. Delio Vicini and Tizian Zeltner were responsible for volume rendering and polarization support respectively. Sébastien Speierer contributed to many features of the system, as well as subsequent development.

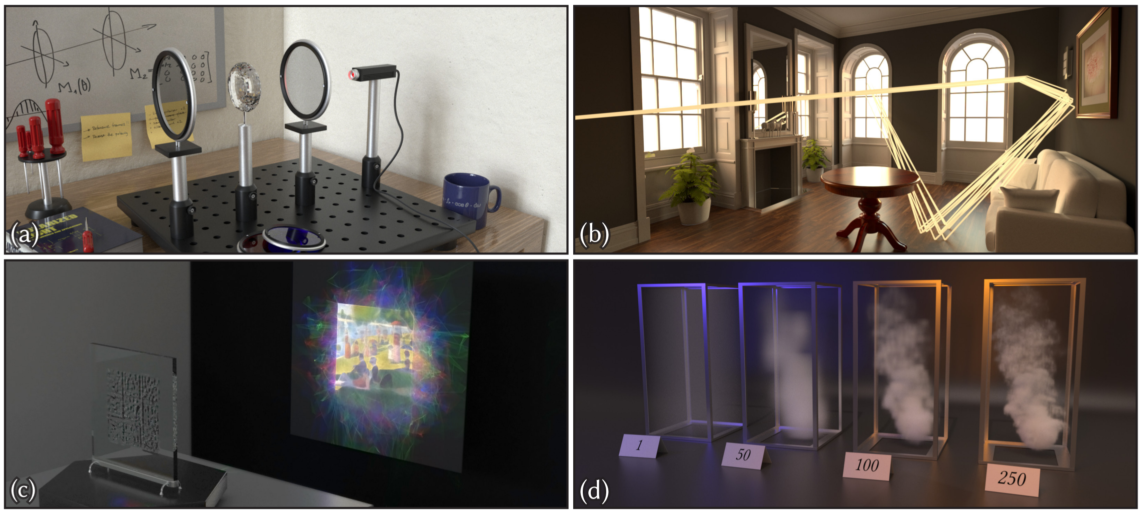


Figure 7.1: Four applications enabled by Mitsuba 2's retargetable architecture. **(a)** Polarized spectral rendering of an optical experiment that analyzes light with elliptical polarization. **(b)** A coherent MCMC rendering algorithm that explores bundles of nearby light paths to improve convergence at equal render time. **(c)** A refractive slab optimized by inverse rendering to focus light with three primary colors into a rendition of the painting *A Sunday Afternoon on the Island of La Grande Jatte* by Georges Seurat. **(d)** Reconstruction of a smoke volume from reference images; multiple iterations of the optimization are shown. Applications **(a)** and **(d)** were investigated by Tizian Zeltner and Delio Vicini respectively.

7.1 Introduction

In the previous chapter, we have discussed the complexities inherent to both primal and differentiable physically based rendering, which impose severe constraints on rendering systems. We now propose Mitsuba 2, an open-source architecture for constructing renderers that are intrinsically retargetable to these application domains. Mitsuba 2 takes generic implementations of a set of standard components (*e.g.* rendering algorithms, BSDF models, etc.), and *lifts* them onto a concrete set of types, systematically transforming the underlying algorithms to enable a particular feature. Possible transformations include changing the representation of radiance (*e.g.* to polarized spectra), generating a "wide" renderer that operates on bundles of light paths using AVX512 vector instructions or CUDA kernels, and automatic differentiation of the entire simulation. These transformations can be chained, which further enriches the space of algorithms that can be derived from a single generic implementation. Concretely, we propose a versatile framework of composable types that exploit compile-time computation to retarget a complete rendering system from a generic specification to concrete implementations suited to a range of different tasks.

Novel applications enabled by our system are illustrated in Figure 7.1. Chapter 9 describes our method for creating gradient-index optics that focus incident illumination into caustics that reproduce multiple user-specified images, as well as heightfield-based caustic design. Chapter 11 leverages Mitsuba 2 to simultaneously recover the lighting and material parameters of indoor scenes from real photographs. The open source implementation of Mitsuba 2 is available at:

`https://github.com/mitsuba-renderer/mitsuba2`

Coherent rendering. The aspects of the system related CPU vectorization are omitted from this thesis for the sake of brevity, please see the original article for details [14]. A Monte Carlo Markov Chain rendering technique that explores bundles of nearby light paths to generate coherent work is additionally proposed.

7.2 Background

We start by reviewing the relevant technical background.

7.2.1 Template metaprogramming

The term *metaprogramming* refers to a broad range of techniques, in which a program is able to rewrite its own structure (or that of another program) either statically at compile time or dynamically during execution. *Template metaprogramming* (TMP) denotes a static variant of this approach that was "discovered" in the early 1990s when it was found that C++ templates could be used to perform Turing-complete computation during compilation [160]. Initially considered a dangerous feature due to generally fragile support and superlinear complexity of template expansion in early compilers, TMP has seen significant refinements and extensions in later revisions to the standard (C++11, 14, 17) that have elevated its status to that of a top-level language feature.

Mitsuba 2 performs nontrivial transformations of complete programs that would generally require custom compiler infrastructure or tools for source code synthesis (in particular, our requirements far exceed the capabilities of "generics" or "macros" available in languages such as Ada, Rust, and .NET). Even within C++, the specifics of our design have only become possible due to extensions in the 2017 standard revision. For this reason, we briefly review relevant features, and how they are used by our system.

Note that TMP usage mainly occurs within the internals of our system—rudimentary familiarity with template concepts suffices when developing Mitsuba 2 code.

- *Templates.* Mitsuba 2 components are specified as generic C++ functions and structures parameterized by unknown target-specific types (*e.g.* the representation of colors or floating point values) and / or constants (*e.g.* size or depth). For instance, the fragment

  ```
  template <typename Float> struct MicrofacetDistribution {
      using Vector3f = Array<Float, 3>; /* ... */
  };
  ```

  declares a data structure parameterized by an arithmetic type (`typename Float`) that is then used in the declaration of a more complex 3D vector type. In the simplest case, `Float` could be an ordinary floating point value. More advanced usage might involve types that perform arithmetic symbolically.

- *Variadic templates* are templates that accept an arbitrary number of arguments. For example, the generic function

  ```
  template <typename... T> auto f(const T&... args) {
      return g(h(args)...);
  }
  ```

  rewrites function invocations of the form `f(x1, x2, ...)` into `g(h(x1), h(x2), ...)`. We use variadic templates to realize virtual method calls (*e.g.* BRDF sampling) on the various targets.

- *Compile-time conditionals* facilitate targeted removal of code fragments subject to user-specified conditions. For instance, suppose that the nested block `/* ...*/` in the snippet

  ```
  if constexpr (is_polarized_v<Spectrum>) { /* ... */ }
  ```

  is only meaningful when dealing with polarized spectra, generating a compilation error in the unpolarized case. To avoid this problem, the `if constexpr` statement queries a *type trait* at compile time, excising the nested block in the negative case.

- *Type computation.* It is often difficult or impossible to define types of expressions in a generic program. To address this flaw, modern C++ constructs enable type specifications that are themselves the result of a compile-time calculation. For instance, the snippet

```
A a;
B b;
using Value = decltype(a[0] + b[0]);
constexpr int Size = A::Size + B::Size;
Array<Value, Size> result = /* ... */;
```

  computes the type resulting from the concatenation of arrays `a` and `b`, while applying standard promotion rules (combining `int` and `float` yields `float`, etc.). Here, `constexpr` denotes computation to be performed at compile time, and `decltype` returns the type of a nested expression without evaluating it.

7.2.2 Expression templates

Widely used numerical libraries such as Eigen [161] and Adept [162] rely on a technique known as *expression templates* (ET) [163]. Mathematical expressions in these frameworks return complex types that encode the sequence of operations needed for evaluation rather than triggering evaluation immediately. This enables global optimizations that would be unavailable when considering the operations individually.

We experimented with expression templates during the early stages of this project but ultimately found them not to be a good fit for Mitsuba 2. The approach is ideal for compact statements (*e.g.* simple matrix updates in the case of Eigen) but does not scale to large expressions that encode complex functionality (*e.g.* a complete microfacet model with visible normal sampling). Because ET cannot model variable reuse and common subexpressions, the size of the expression templates tends to grow exponentially as a function of the size of the program, which eventually prevents practical usage.

7.2.3 The Enoki library

Mitsuba 2 is built onto the *Enoki* library of Jakob [13]. For completeness and since its components and interface are highly relevant to the design of Mitsuba 2, we review them here—please see the original article for full detail [14]. Enoki is a template library responsible for vectorization, JIT compilation, and program transformations. Intended to be as general as possible, it does not contain any rendering-specific code. Its main components are:

- A lazy *just-in-time* (JIT) compiler that symbolically executes arithmetic and control flow to generate computational kernels for later execution on a GPU.
- An efficient graph-based approach for simultaneous forward- and reverse-mode *automatic differentiation* (AD) that seamlessly composes with other transformations.
- A simplification algorithm that periodically simplifies the graph data structure used by automatic differentiation to reduce the memory usage of differentiable rendering.

Static arrays. The fundamental building block of Enoki is a generic fixed-size container `Array<Value, Size>`, whose default implementation intercepts and carries out arithmetic operations component-wise by forwarding them to its elements (for instance, `a = b + c` will be rewritten into `a[i] = b[i] + c[i]`). Its template parameter `Value` could be an arbitrary data structure (*e.g.* a string), an arithmetic type, or another Enoki array. These containers can be arbitrarily nested to create higher-order tensors, such as a $4 \times 4 \times 8 \times N$ array containing a packet of N spectral Mueller matrices that are each sampled at 8 discrete wavelengths.

All arrays furthermore support implicit *broadcasting*, which suitably expands the size of a tensor so that an operation can be carried out. This is particularly important where vectorized and non-vectorized portions of the system meet. For instance, suppose that the aforementioned higher-order tensor occurs in a product involving another Mueller matrix or a discrete color spectrum. In such a case, Enoki inspects the types of the involved operands at compile time to determine that a broadcast to dimensions (1,2) or 3 of the rank-4 tensor is necessary.

Enoki provides *vertical* and *horizontal* arithmetic operations that are each split into a target-independent *frontend* portion responsible for broadcasting and type conversion, and a target-specific *backend* portion. Vertical operations proceed component-wise and produce a tensor of the same shape, while horizontal operations involve a reduction over one or more dimensions, returning a tensor of lower rank.

Frontend. The frontend part takes two arguments of type `T1` and `T2`, of which at least one must be an Enoki array.

```
template <typename T1, typename T2, enable_if_array_t<T1, T2> = 0>
auto operator*(const T1 &a1, const T2 &a2) {
    using E = expr_t<T1, T2>;
    if constexpr (is_same_v<T1, E> && is_same_v<T2, E>)
        return a1.mul_(a2);
    else
        return operator*(E(a1), E(a2));
}
```

The expression `expr_t<T1, T2>` uses TMP to compute the type `E` of an expression involving `a1` and `a2`, while accounting for steps such as type promotion and broadcasting. This step is repeated recursively, until `T1`, `T2`, and `E` are all identical, at which point the operation is forwarded to the backend method `mul_()`.

CPU Backend. The default backend executes the operation on the desired target platform, relying on a pattern matching mechanism known as *partial specialization*. A large set of `Array<Value, Size>` specializations intercept certain combinations of `Value` and `Size` that are natively supported. The pattern matching mechanism is recursive—arrays with too large or odd sizes that prevent vectorization are partitioned into two sub-arrays, whose larger part is a power of two, and the process repeats anew. This all happens during compilation and hence incurs no runtime cost. At the time of writing, Enoki provided backends for SSE4.2, AVX, AVX2, and AVX512 on Intel-compatible processors, NEON on ARM processors, and a scalar fallback mode. Unlike manually vectorized code that relies on compiler intrinsics, the combination of routing and partial specialization makes algorithms developed in Enoki platform-independent (a similar goal is pursued by ISPC [71]).

GPU backend (lazy JIT compiler). Another array type, `GPUArray<Value>`, provides *dynamically sized* 1D arrays that are stored on a graphics card. A simple backend for such an array could dispatch each arithmetic operation to a pre-compiled GPU kernel[1], but this leads to poor hardware utilization due to memory traffic (repeated reads and writes of operands) and the large overhead of launching kernels for such a small amount of computation. Consider the following simple program, which counts how many elements of a randomly distributed set of points on $[0, 1]^3$ fall within a sphere of radius one:

[1]This is in fact what frameworks such as PyTorch do by default.

```
1 using Float    = GPUArray<float>;
2 using UInt64   = GPUArray<uint64_t>;
3 using Vector3f = Array<Float, 3>;
4 PCG32<UInt64> rng(arange<UInt64>(1000000));
5 Vector3f v(rng.next_float(), rng.next_float(), rng.next_float());
6 size_t inside = count(norm(v) < 1.f);
```

Here, lines 1-3 set up the necessary types, `arange<UInt64>()` in line 4 generates an integer sequence with 1 million entries to select separate streams of a PCG32 random number generator [22], and line 6 carries out a horizontal counting operation. PCG32 is a linear congruential generator, and operations involving it reduce to a sequence of multiplications and bit-level manipulations (XOR, OR, shifts, etc.).

This example is extremely simple compared to a typical physically based shading model, yet over 180 kernel launches would be needed to execute it using the previously mentioned approach (56 for seeding the random number generators, 32 for each array of samples, and 7 for the final count). While we could create specialized kernels that combine some of these operations (*e.g.* to generate uniform variates), this approach clearly does not scale to the complexities of an entire renderer.

Enoki's solution to this problem is to perform arithmetic *symbolically*: the backend merely records the desired sequence of operations, postponing evaluation of the kernel for as long as possible. Only once we "peek" inside an array (*e.g.* in line 6 of the previous example) is it necessary to actually compute its contents. The GPU backend exploits this using a lazy tracing *just-in-time* (JIT) compilation approach. It outputs NVIDIA's *Parallel Thread Execution* (PTX) intermediate representation to construct a program in *single static assignment* (SSA) form.

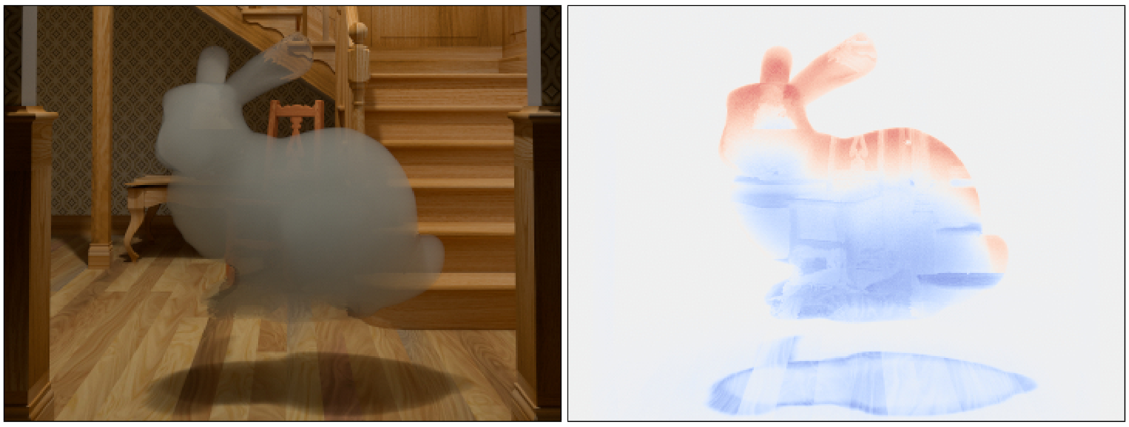
A subsequent compiler pass is then necessary to generate an executable kernel using the native GPU instruction set SASS. The same pass also performs register allocation and optimizations, such as common subexpression elimination and constant folding. Since this is by far the costliest part of JIT compilation, the resulting kernels are cached and reused if the same computation occurs again. This particularly helps when running an iterative algorithm like stochastic gradient descent, in which case compilation typically only needs to occur once during the first iteration.

Autodiff backend. Enoki's last array type, `DiffArray<Value>`, enables transparent forward and reverse-mode differentiation. Similar to `autograd` in PyTorch [62] or Stan [164], gradient evaluation requires a declaration of relevant inputs followed by a statement that triggers the graph traversal. Example usage is shown in Listing 7.

```
// Forward-mode AD:
using Float = DiffArray<float>;
Float in = 1.0f;
set_requires_gradient(in);

auto [out1, out2] = f(in);
forward(in);
float grad1 = gradient(out1);
float grad2 = gradient(out2);
```

```
// Reverse-mode AD:
using Float = DiffArray<float>;
Float in1 = 1.f, in2 = 2.f;
set_requires_gradient(in1);
set_requires_gradient(in2);
Float out = f(in1, in2);
backward(out);
float grad1 = gradient(in1);
float grad2 = gradient(in2);
```

Listing 7: Example usage of forward- and reverse-mode automatic differentiation with Enoki.

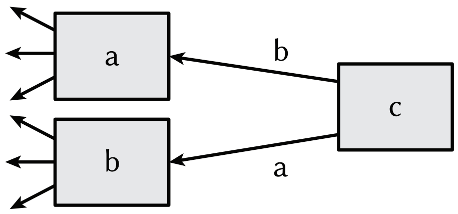


(a) Color image (b) Gradient image

Figure 7.2: Visualization of the gradient of an image with respect to the density of a participating medium, computed using forward-mode automatic differentiation (red and blue encode positive and negative values, respectively). Figure by Delio Vicini.

Both forward and reverse-mode AD have useful applications in the context of rendering: the former to visualize gradients for a scene parameter (Figure 7.2), and the latter to optimize a scene with respect to an objective function involving a rendered image (Figure 7.1 (c, d)).

A `DiffArray` consists of two parts: a value that is used during the forward pass, and a reference to a node in a separately maintained directed acyclic graph capturing the structure of the computation. For example, for the multiplication `c = a * b`, the backend creates a new node `c` referencing the operands:



Edges are *weighted* and store partial derivatives of the operation with respect to its inputs—here, this is simply the product rule. Reverse- or forward-mode traversal entails a sequence of multiply-accumulate operations to apply the chain rule. A reverse-mode traversal of the above graph triggers two updates: `da += b * dc` and `db += a * dc`.

Automatic differentiation on the GPU. In practice, we generally combine the previously described array types by nesting them, making `DiffArray<GPUArray<float>>` the basic "numeric" type for Mitsuba 2's differentiable rendering mode. The combination of a lazy JIT compiler with AD has interesting consequences: computation related to derivatives is queued up along with primal arithmetic and can thus be compiled into a joint GPU kernel[2], leveraging subexpression elimination and constant folding to further improve efficiency.

Masks. Vectorized algorithms process multiple elements at once, hence standard language features like `if` statements are unsuitable for modeling their control flow[3]. Comparisons and other logical operations involving Enoki arrays thus produce *masks* (arrays of boolean values). Masks are often used to select from one of two expressions using the ternary conditional operator `select(mask, expr_true, expr_false)`.

Horizontal operations. Horizontal operations involve a reduction over one or more dimensions, returning a tensor of lower rank. Examples are logical reductions that can be applied to masks (`all()`, `any()`, `none()`, `count()`, etc.) and arithmetic reductions for standard arrays (horizontal sums, products, maxima, etc.). Their realization depends on the target.

When working with GPU arrays, horizontal reductions are best avoided whenever possible. Vertical operations are scheduled asynchronously and execute concurrently

[2]For example, if a forward computation evaluates the expression `sin(x)`, the weight of the associated backward edge in the computation graph is given by `cos(x)`. The computation of both of these quantities is automatically merged into a single joint kernel.

[3]A compiler frontend like ISPC [71] has an advantage here, since it can automate the conversion of conditional statements to masks.

on the entire chip, which is key to their efficiency. In contrast, horizontal operations create synchronization barriers that require all queued computation to finish before the reduction can take place. They are also often unnecessary: for example, `any(mask)` is almost certainly `true` if `mask` is a large array and the underlying condition is satisfied with a nonzero probability. For this reason, Enoki provides logical reductions with a default choice (*e.g.* `all_or<true>`) that takes precedence when targeting the GPU. Integrating these improvements into the previous example addresses the discussed flaws.

Method dispatch. Indirect branches are a common feature of rendering code, for example to sample the BSDF of an intersected shape:

```
SurfaceInteraction3f si = /* ... */;
BSDFSample3f bs = si.bsdf->sample(si, sample);
```

In a scalar program, this operation represents an ordinary virtual function call that requires no special handling. Vectorization, however, turns `si.bsdf` into an array of pointers, that potentially refer to many different BSDF instances. Enoki intercepts such function calls by overloading the "->" operator and dispatches them using one method call per *unique* pointer.

The details of this step vary depending on the target. On the GPU, Enoki issues a horizontal operation to extract a list of unique pointers along with a list of indices per pointer specifying what elements of the array refer to it. Following this, the argument values associated with a particular instance are gathered, the function call is performed, and results are scattered back into the output array. This is roughly equivalent to the following pseudocode:

```
for (bsdf, indices) in partition(si.bsdf):
    BSDFSample3f temp = bsdf->sample(gather(si, indices),
                                     gather(sample, indices))
    scatter(bs, temp, indices)
```

All three steps merely enqueue computation to be executed at a later point. Enoki uses TMP to automatically rewrite the earlier virtual function call into this form, hence no target-specific code is necessary. For differentiable arrays, Enoki automatically propagates derivative information through virtual function calls.

Other features. Enoki's autodiff backend includes an automatic graph simplification algorithm, which periodically merges nodes and edges to reduce storage requirements. This is crucial to the system's scalability when attempting to perform inverse rendering optimizations beyond trivial rendering resolution or scene complexity. A broad range of mathematical functions, expressed using Enoki's array types, is also provided. For further details, please see the original article [14].

7.3 System design

We now turn to the design of Mitsuba 2 itself, which was influenced by three guiding principles:

- *No duplication.* Special cases will inevitably arise during certain program transformations, we wish to support these without creating many special variants of an algorithm.
- *Unobtrusiveness.* Several transformations discussed in Section 7.1 substantially increase the size and complexity of an implementation, obscuring physical and algorithmic concepts. We thus want development to take place at the *input end* of such transformations. The development of generic algorithms should furthermore resemble their "classical" counterparts as much as possible.
- *Modularity.* Physically based rendering systems admit a particularly modular architecture and are often partitioned into a large set of loadable plug-in modules that implement materials, rendering algorithms, and so on. To support the same level of modularity, our approach should be compatible with separate compilation of the various parts of the renderer.

As mentioned in Section 7.2.3, our system builds on top of Enoki. It implements a complete rendering system designed for compatibility with the Mitsuba 0.6 [26] scene description language, and replicates Mitsuba's interfaces and plugins with suitable abstractions that admit the discussed program transformations. Mitsuba 2 also provides fine-grained bindings and utilities for prototyping forward and inverse rendering pipelines in Python. We now examine the system's architecture in more detail.

7.3.1 Architecture

Figure 7.3 illustrates the relationship of Mitsuba 2 (top, green) to Enoki (bottom, beige). As described in Section 7.2.3, Enoki provides `Array` types that can be manipulated uniformly, as they expose the same interface. Operations on Enoki arrays are routed to

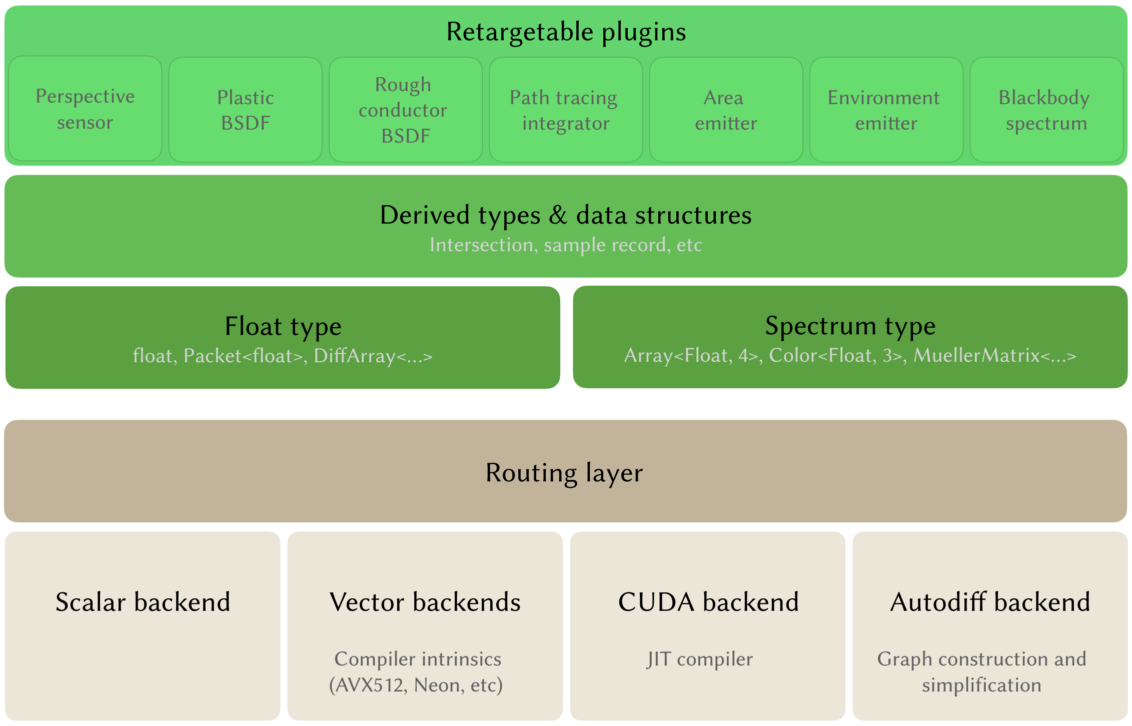


Figure 7.3: Mitsuba 2's components (top, green) are written as templates, which are then instantiated using Enoki types (bottom, beige) in order to automatically perform the supported program transformations.

various backends that efficiently realize arithmetic operations, vertical and horizontal reductions, and virtual function calls.

Template parameters. In turn, the majority of Mitsuba 2's components are written as C++ *templates*, taking types `Float` and `Spectrum` as parameters. Selecting a particular combination of `Float` and `Spectrum` types effectively *retargets* the entire system to use the desired backend and color handling. Typical values for the template types include:

- `Float`: `float` or `double` for standard scalar operation, `Packet<float>` for CPU-based vectorization (SIMD intrinsics), and `CUDAArray<float>` for JIT compilation to CUDA kernels. Autodiff is enabled by further wrapping the `Float` type, *e.g.* `DiffArray<CUDAArray<float>>`.

- `Spectrum`: `Color3f` for RGB rendering, `Spectrum<Float, 4>` for spectral color handling with 4 spectral samples per ray, and `MuellerMatrix<Spectrum<Float, 4>>` for polarized light simulation.

Data structures. Modern renderers use auxiliary data structures to facilitate communication between different system components. This includes surface and medium interactions, data structures for direct illumination and BSDF sampling, and so on. In Mitsuba 2, these are declared as templates in order to permit retargeting. When applicable, we template over a higher-level type such as `Point`, so that the data structure can be retargeted in one more way: the dimensionality of world space (*e.g.* 2D, 3D or 4D). For example, the renderer's surface intersection data structure roughly looks as shown in Listing 8.

In the structure's declaration, all relevant types from 3D vectors to arrays of pointers, are then derived from the template parameters using Enoki's helper type traits. `value_t<T>` extracts the value underlying an array `T`, and `replace_scalar_t<T, X>` returns an array of the same structure as `T`, but using a representation based on the (scalar) type `X`. For instance, `uint32_array_t<T>` is an alias for `replace_scalar_t<T, uint32_t>` and returns an unsigned integer version of the argument. The macro `ENOKI_STRUCT` allows certain Enoki operations to be applied to the data structure itself, causing them to recursively propagate through all fields.

```
template <typename Point3f> struct SurfaceInteraction {
    using Float    = value_t<Point3f>;
    using Vector3f = Vector<Float, 3>;
    using Frame3f  = Frame<Vector3f>;
    using UInt32   = uint32_array_t<Float>;
    using Shape3f  = replace_scalar_t<Float, const Shape<Float> *>;

    Float t;          // ray distance
    Point3f p;        // position
    Vector3f wi;      // incident direction
    Frame3f sh_frame; // shading coordinate frame
    UInt32 prim_id;   // intersected primitive (e.g. triangle ID)
    Shape3f shape;    // pointer to Shape<..> instance
    /// ...
};
ENOKI_STRUCT(SurfaceInteraction, t, p, wi, sh_frame, prim_id, shape)
```

Listing 8: Excerpt from Mitsuba 2's `SurfaceInteraction` data structure. Its fields are typed using Enoki array types, derived from the structure's template type parameter.

Structure of arrays. This type of recursive lifting greatly facilitates tasks such as switching data structures to a *Structure of Arrays* (SoA) representation. For this, we can simply substitute a vectorized floating point type at the root level (*e.g.* `Array<float,`

16> or GPUArray<**float**>), letting the type system do the remaining work. Figure 7.4 illustrates the resulting memory layout for two example values of the type Float.

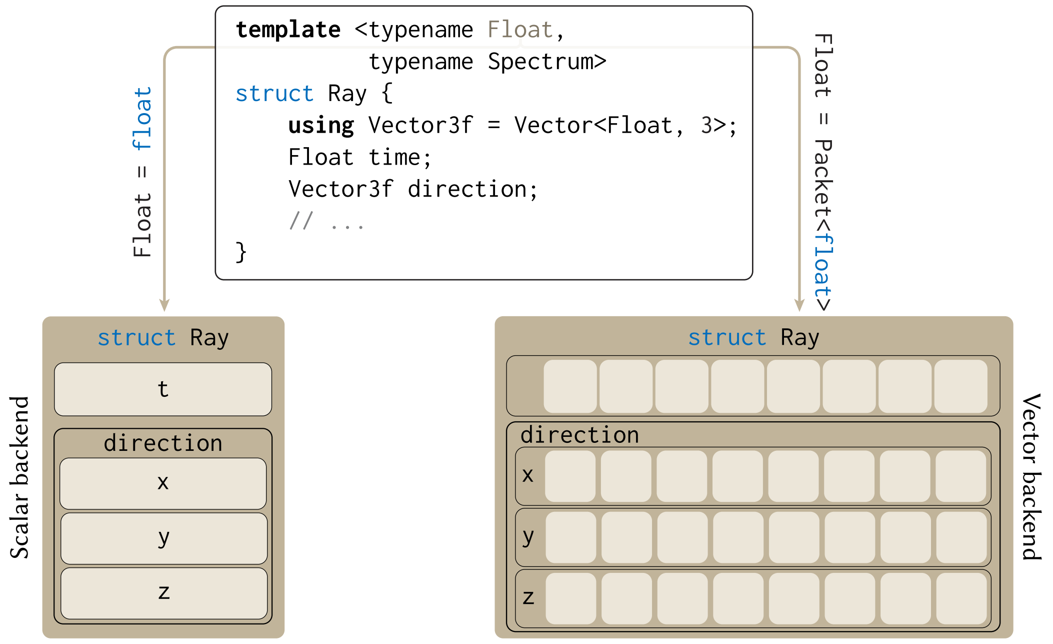
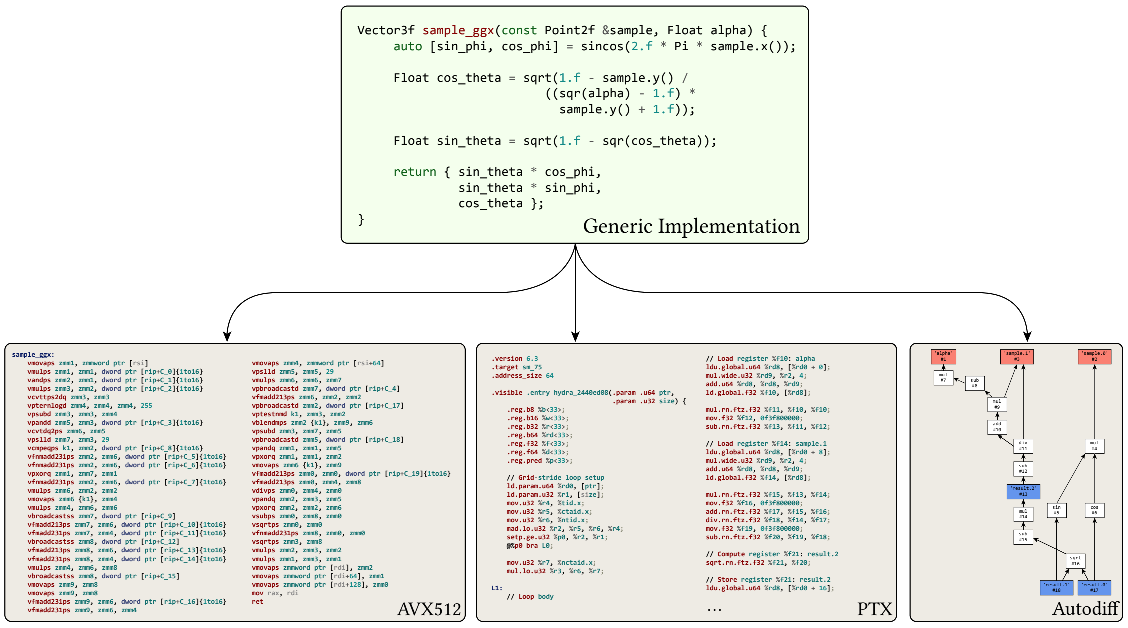


Figure 7.4: Excerpt of Mitsuba 2's Ray data structure, which is written as a generic **struct**. Depending on the Float and Spectrum types, it will yield a standard scalar structure, a vectorized structure-of-arrays (SoA) for use with SIMD instructions, a dynamically-sized SoA located in GPU memory, etc.

Plugins. Interfaces describing the (virtual) methods and attributes of all major scene components, such as Emitter, Shape and Sensor are once again implemented as template classes. Finally, plugins inheriting from the interfaces are implemented in individual compilation units. The type parameters are passed to the parent interface, for example:

```
template <typename Float, typename Spectrum>
class AreaEmitter final : public Emitter<Float, Spectrum> {
    // ...
}
MTS_EXPORT_PLUGIN(AreaEmitter, "Area light source")
```

The MTS_EXPORT_PLUGIN macro is responsible for explicitly instantiating the templates for all combinations of the Float and Spectrum types that are currently enabled by the compilation options. A larger excerpt from a plugin implementation is given in Listing 10, and Figure 7.5 shows an example method's instantiation to different backends.



Generic Implementation

```
Vector3f sample_ggx(const Point2f &sample, Float alpha) {
    auto [sin_phi, cos_phi] = sincos(2.f * Pi * sample.x());

    Float cos_theta = sqrt(1.f - sample.y() /
                          ((sqr(alpha) - 1.f) *
                            sample.y() + 1.f));

    Float sin_theta = sqrt(1.f - sqr(cos_theta));

    return { sin_theta * cos_phi,
             sin_theta * sin_phi,
             cos_theta };
}
```

AVX512

```
sample_ggx:
    vmovaps zmm1, zmmword ptr [rsi]
    vmulps zmm1, zmm1, dword ptr [rip+C_0]{1to16}
    vandps zmm2, zmm1, dword ptr [rip+C_1]{1to16}
    vmulps zmm3, zmm2, dword ptr [rip+C_2]{1to16}
    vcvttps2dq zmm3, zmm3
    vpternlogd zmm4, zmm4, zmm4, 255
    vpsubd zmm3, zmm3, zmm4
    vpandd zmm5, zmm3, dword ptr [rip+C_3]{1to16}
    vcvtdq2ps zmm6, zmm5
    vpslld zmm7, zmm3, 29
    vcmpeqps k1, zmm2, dword ptr [rip+C_8]{1to16}
    vfnmadd231ps zmm2, zmm6, dword ptr [rip+C_5]{1to16}
    vfnmadd231ps zmm2, zmm6, dword ptr [rip+C_6]{1to16}
    vpxorq zmm1, zmm7, zmm1
    vfnmadd231ps zmm2, zmm6, dword ptr [rip+C_7]{1to16}
    vmulps zmm6, zmm2, zmm2
    vmovaps zmm6 {k1}, zmm4
    vmulps zmm4, zmm6, zmm6
    vbroadcastss zmm7, dword ptr [rip+C_9]
    vfmadd213ps zmm7, zmm6, dword ptr [rip+C_10]{1to16}
    vfmadd231ps zmm7, zmm4, dword ptr [rip+C_11]{1to16}
    vbroadcastss zmm8, dword ptr [rip+C_12]
    vfmadd213ps zmm8, zmm6, dword ptr [rip+C_13]{1to16}
    vfmadd231ps zmm8, zmm4, dword ptr [rip+C_14]{1to16}
    vmulps zmm4, zmm6, zmm8
    vbroadcastss zmm8, dword ptr [rip+C_15]
    vmovaps zmm9, zmm8
    vmovaps zmm9, zmm8
    vfmadd231ps zmm9, zmm6, dword ptr [rip+C_16]{1to16}
    vfmadd231ps zmm9, zmm6, zmm4
```

```
    vmovaps zmm4, zmmword ptr [rsi+64]
    vpslld zmm5, zmm5, 29
    vmulps zmm6, zmm6, zmm7
    vpbroadcastd zmm7, dword ptr [rip+C_4]
    vfmadd213ps zmm6, zmm2, zmm2
    vpbroadcastd zmm2, dword ptr [rip+C_17]
    vptestnmd k1, zmm3, zmm2
    vblendmps zmm2 {k1}, zmm9, zmm6
    vpsubd zmm3, zmm7, zmm5
    vpbroadcastd zmm5, dword ptr [rip+C_18]
    vpandq zmm1, zmm1, zmm5
    vpxorq zmm1, zmm1, zmm2
    vmovaps zmm6 {k1}, zmm9
    vfmadd213ps zmm0, zmm0, dword ptr [rip+C_19]{1to16}
    vfmadd213ps zmm0, zmm4, zmm8
    vdivps zmm0, zmm4, zmm0
    vpandq zmm2, zmm3, zmm5
    vpxorq zmm2, zmm2, zmm6
    vsubps zmm0, zmm8, zmm0
    vsqrtps zmm0, zmm0
    vfnmadd231ps zmm8, zmm0, zmm0
    vsqrtps zmm3, zmm8
    vmulps zmm2, zmm3, zmm2
    vmulps zmm1, zmm3, zmm1
    vmovaps zmmword ptr [rdi], zmm2
    vmovaps zmmword ptr [rdi+64], zmm1
    vmovaps zmmword ptr [rdi+128], zmm0
    mov rax, rdi
    ret
```

PTX

```
.version 6.3
.target sm_75
.address_size 64

.visible .entry hydra_2440ed08(.param .u64 ptr,
                               .param .u32 size) {
    .reg.b8 %b<33>;
    .reg.b16 %w<33>;
    .reg.b32 %r<33>;
    .reg.b64 %rd<33>;
    .reg.f32 %f<33>;
    .reg.f64 %d<33>;
    .reg.pred %p<33>;

    // Grid-stride loop setup
    ld.param.u64 %rd0, [ptr];
    ld.param.u32 %r1, [size];
    mov.u32 %r4, %tid.x;
    mov.u32 %r5, %ctaid.x;
    mov.u32 %r6, %ntid.x;
    mad.lo.u32 %r2, %r5, %r6, %r4;
    setp.ge.u32 %p0, %r2, %r1;
    @%p0 bra L0;

    mov.u32 %r7, %nctaid.x;
    mul.lo.u32 %r3, %r6, %r7;

L1:
    // Loop body
```

```
    // Load register %f10: alpha
    ldu.global.u64 %rd8, [%rd0 + 0];
    mul.wide.u32 %rd9, %r2, 4;
    add.u64 %rd8, %rd8, %rd9;
    ld.global.f32 %f10, [%rd8];

    mul.rn.ftz.f32 %f11, %f10, %f10;
    mov.f32 %f12, 0f3f800000;
    sub.rn.ftz.f32 %f13, %f11, %f12;

    // Load register %f14: sample.1
    ldu.global.u64 %rd8, [%rd0 + 8];
    mul.wide.u32 %rd9, %r2, 4;
    add.u64 %rd8, %rd8, %rd9;
    ld.global.f32 %f14, [%rd8];

    mul.rn.ftz.f32 %f15, %f13, %f14;
    mov.f32 %f16, 0f3f800000;
    add.rn.ftz.f32 %f17, %f15, %f16;
    div.rn.ftz.f32 %f18, %f14, %f17;
    mov.f32 %f19, 0f3f800000;
    sub.rn.ftz.f32 %f20, %f19, %f18;

    // Compute register %f21: result.2
    sqrt.rn.ftz.f32 %f21, %f20;

    // Store register %f21: result.2
    ldu.global.u64 %rd8, [%rd0 + 16];

    ...
```

Figure 7.5: Starting from an algorithmic template (shown: importance sampling a GGX lobe), our system generates high-quality vectorized implementations for CPUs and GPUs. Further instrumentation in Enoki tracks the computation graph and enables forward and reverse-mode automatic differentiation.

Variants. At runtime, users start by selecting a *variant*, *i.e.* a combination of `Float` and `Spectrum` types with which to instantiate the renderer. This selection can be made using a C++ helper macro which switches over all available variants, or more conveniently through the Python bindings. The variant must have been enabled at compile time to be available at runtime. Once a variant has been selected, instantiating a scene will dynamically load the corresponding implementation or each plugin used in the scene.

A number of utility classes and functions, such as the `Bitmap` and `Thread` classes, are not implemented as templates: their functionality, *e.g.* performing I/O operations, should be the same regardless of the current variant.

7.3.2 Language bindings

Mitsuba 2's focus on types has another less obvious benefit: they provide a rich description of structures and memory layout that enables high-quality language bindings. We extended `pybind11` [8]—itself based on metaprogramming—to create bindings from one-line declarations of the form

```
module.def("func", &func);
```

A metaprogram then analyzes the function's type to synthesize code that automatically converts function arguments and return values. We used this approach to create fine-grained Python language bindings of all major rendering system components for CPU (scalar and vectorized) and GPU (vectorized differentiable) targets, enabling prototyping of complete rendering algorithms, *e.g.* using interactive Jupyter notebook sessions. Enoki arrays also support implicit bidirectional conversion to other array libraries, such as NumPy and PyTorch. The latter allows the renderer to be used as a differentiable layer in a larger computation graph realized using PyTorch.

In fact, the optimization pipelines of Chapters 9 and 11 were all developed in Python scripts that begin by loading an XML scene that specifies the starting point of the optimization. The scene can be queried for differentiable parameters, some of which are subsequently connected to an optimizer (SGD, Adam, etc.) along with standard or custom loss functions. A short example of forward rendering via Python is shown in Listing 9.

```python
import mitsuba
mitsuba.set_variant('scalar_rgb')
from mitsuba.core.xml import load_file, load_dict

scene = load_file('cbox/scene.xml')
integrator = load_dict({
    'type': 'path',
    'max_depth': 64,
})
image = integrator.render(scene)
```

Listing 9: Example usage of Mitsuba 2's features through the Python language bindings. Almost all types and functions are exposed, enabling rapid development of primal and inverse rendering algorithms.

7.3.3 Feature set

We used the abstractions of Enoki to implement a complete rendering system that includes (at the time of writing) a path tracer, volumetric path tracer, and adjoint light tracer (all with multiple importance sampling). Our system supports standard light sources (point, area, directional lights and environment maps) and both specular and rough microfacet BRDFs with visible normal sampling [165] for conductors, dielectrics, and plastic-like materials. As described above, each component of the renderer is compiled as a plugin (*i.e.* a shared library) that contains multiple instantiations of the template (abstract implementation).

The renderer is spectral by default and uses Monte Carlo sampling to integrate over continuous wavelengths spanning the 360 to 830nm range using 4 sampled wavelengths per ray. A monochromatic mode is also available, which is mainly used in automated tests, and for debugging. The transformation from RGB values (*e.g.* in texture maps) to reflectance spectra relies on the vectorized spectral upsampling model of Jakob and Hanika [29].

Three backends are available for ray tracing: scalar and packet tracing on the CPU either use a built-in kd-tree or Embree [166], and ray tracing on the GPU relies on OptiX [167]. The built-in kd-tree is useful for debugging, *e.g.* to render an image in double precision which neither Embree nor OptiX support.

7.3.4 Challenges

During development, we encountered two standard constructions that require special precautions. First, sampling code often relies on Newton or Newton-Bisection iterations to numerically invert cumulative distribution functions (CDFs), whose inverse does not have a closed-form expression. The iteration's stopping criterion `if (all(converged))` poorly interacts with the GPU backend, since this is a horizontal operation that would serialize the computation at every step. In such cases, we determined[4] suitable fixed upper bounds for the iteration count that we use instead. A related example are discrete CDFs inverted using a binary search, *e.g.* to pick rows and columns of an environment map. Here, a tight bound is given by $\lceil \log_2(N-2) \rceil + 1$, where N is the number of entries. Following this change, all Newton iterations or binary search steps are unrolled into the current kernel.

For gradient-based optimization, we had to ensure that certain operations that normally run as a pre-processing step before rendering begins are recorded in the computation graph. An example is the computation of smooth shading normals from vertex positions. One is a function of the other, hence it is important to accurately capture their relationship during optimization. Our system provides reconstruction filters (*e.g.* a Gaussian or Mitchell-Netravali filter), whose contribution to the image is differentiable with respect to the position of a sample. This is necessary *e.g.* to optimize the shape of caustics due to chains of purely specular transport, which will be the focus of Chapter 9.

[4]There are "only" 4 billion single precision floating point values, and it is normally possible to test all of them in a few minutes.

7.4 Evaluation

We delay evaluation of the differentiable aspects of the system to the inverse rendering applications of Chapters 9 and 11. Please refer to the original article [14] and its supplemental material for a more complete evaluation.

Example plugin implementation. For qualitative evaluation, Listing 10 shows an excerpt from the standard diffuse BSDF model implemented in Mitsuba 2. Beyond the template parameters and a small number of macros, the implementation looks strikingly similar to standard C++ physically based rendering code, *e.g.* from Mitsuba 0.6 [26]. The complexity is successfully encapsulated in the type system and the Enoki library.

Comparison to Redner. The performance of our JIT-compiled kernels for differentiable rendering is competitive with hand-written derivative code: in particular, we found, at the time of writing, that Mitsuba 2 was typically faster than *Redner*, the open source implementation of the method by Li et al. [57] (Table 7.1).

Table 7.1: Timing comparison against *Redner* when optimizing diffuse appearance model parameters. Special handling of discontinuities was disabled in both renderers to only benchmark the differentiable parts of the problem. Additional information on this comparison can be found in the supplemental material of the original article [14]. For benchmarks that include the effects of visibility, please refer to the article of Loubet et al. [112].

	Number of Parameters	Redner	Mitsuba 2
Cornell box	5×3	0.3553 s/it	0.3140 s/it
Textured monkey	$512 \times 512 \times 3$	0.2280 s/it	0.1501 s/it
Textured monkey	$1024 \times 1024 \times 3$	0.2293 s/it	0.1503 s/it
Textured sphere	$512 \times 512 \times 3$	0.1991 s/it	0.1132 s/it
Textured sphere	$1024 \times 1024 \times 3$	0.1998 s/it	0.1133 s/it

7.5 Conclusion

Physically based rendering is the result of a complex interplay involving countless different system components. Similar to how a photon can interact with distant parts of a large and detailed scene, program execution in a renderer tends to take twisty paths through immense codebases, whose size is measured in multiple hundred thousand lines

```cpp
template <typename Float, typename Spectrum>
class SmoothDiffuse final : public BSDF<Float, Spectrum> {
public:
    MTS_IMPORT_BASE(BSDF, m_flags, m_components)

    SmoothDiffuse(const Properties &props) : Base(props) {
        m_reflectance = props.texture<Texture>("reflectance", .5f);
        // ...
    }

    Spectrum eval(const BSDFContext &ctx, const SurfaceInteraction3f &si,
                const Vector3f &wo, Mask active) const override {
        // ...
        Float cos_theta_i = Frame3f::cos_theta(si.wi),
              cos_theta_o = Frame3f::cos_theta(wo);
        active &= cos_theta_i > 0.f && cos_theta_o > 0.f;

        UnpolarizedSpectrum value = m_reflectance->eval(si, active)
                                    * math::InvPi<Float> * cos_theta_o;
        return select(active, unpolarized<Spectrum>(value), 0.f);
    }
    // ...
    MTS_DECLARE_CLASS()
private:
    ref<Texture> m_reflectance;
};

MTS_IMPLEMENT_CLASS_VARIANT(SmoothDiffuse, BSDF)
MTS_EXPORT_PLUGIN(SmoothDiffuse, "Smooth diffuse material")
```

Listing 10: Excerpt from the C++ implementation of Mitsuba 2's diffuse BSDF plugin.

of code. But simply rendering an image is often not enough—depending on the application, the entire process needs to be very accurate, very fast, or differentiable (or worse, several of the above). Such requirements imply painstaking global transformations into highly specialized implementations that are challenging to understand and maintain.

These challenges motivate the design of our system: the combination of generic algorithms and composable compile-time transformations of types enable development at a high level of abstraction. Without code duplication, our system is then able to generate high-quality scalar, vector and GPU implementations with competitive performance. Another type of transformation changes the representation of radiance, making light transport effects like polarization considerably easier to support. Finally, Enoki's lazy JIT compiler and automatic differentiation support unlock a path to straightforward conversion of any rendering algorithm or appearance model into an optimization technique for solving associated inverse problems.

Limitations. A number of limitations and open questions remain: due to our reliance on deeply nested templates, error messages provided by the compiler can be cryptic. During development, we address such problems by performing a scalar-only build of the renderer—once this succeeds, the other variants should follow suit, assuming that the transformations themselves are correct. C++20 introduces a feature named *concepts* that will likely address this problem more elegantly. Mitsuba 2 code also requires a conversion of conditional statements into masks, which can be tedious when a model requires intricate conditional logic.

Despite the optimizations pursued in our system and the Enoki library, reverse-mode automatic differentiation significantly increases the amount of application state, and parameters like resolution, sample count, and the number of passes, require careful adjustment to avoid out-of-memory errors. Furthermore, our GPU backend renders images using a sequence of separate kernel launches that exchange information through global memory, which causes large communication-related overheads.

Extensions. Systems like OptiX that compile an entire renderer into a single "megakernel" avoid this type of overhead, although their increased register usage tends to impede the latency-hiding ability of modern GPUs [168]. We currently unroll loops and recursive algorithms, which is likely not always ideal. The automatic translation of the wavefront-style rendering algorithms of Mitsuba 2 into efficient megakernels, including differentiation, is the subject of Chapter 8.

Impact. Mitsuba 2 has already proven to be a helpful tool for researchers in computer graphics and computer vision, being successfully deployed in at least 72 published works at the time of writing (*e.g.* [130, 135, 169]). We believe that its applicability extends to many other areas (*e.g.* design or architecture) that optimize geometry or materials to achieve a goal that can be specified as a differentiable algorithm.

8 | From wavefront to megakernel

Relevant background: Section 2.5, Chapters 4 and 7.

Mitsuba 2, presented in the previous chapter, is a highly flexible architecture for retargetable physically based rendering. Combined with Enoki's autodiff backend, it enables applying automatic differentiation through the entire light transport simulation, including virtual function calls. However, this fully AD-based approach quickly shows its limit as the rendering resolution or scene complexity increases. Indeed, the large memory footprint of the AD graph forces users to split computation into multiple passes, drastically slowing down each iteration of *e.g.* inverse rendering optimizations.

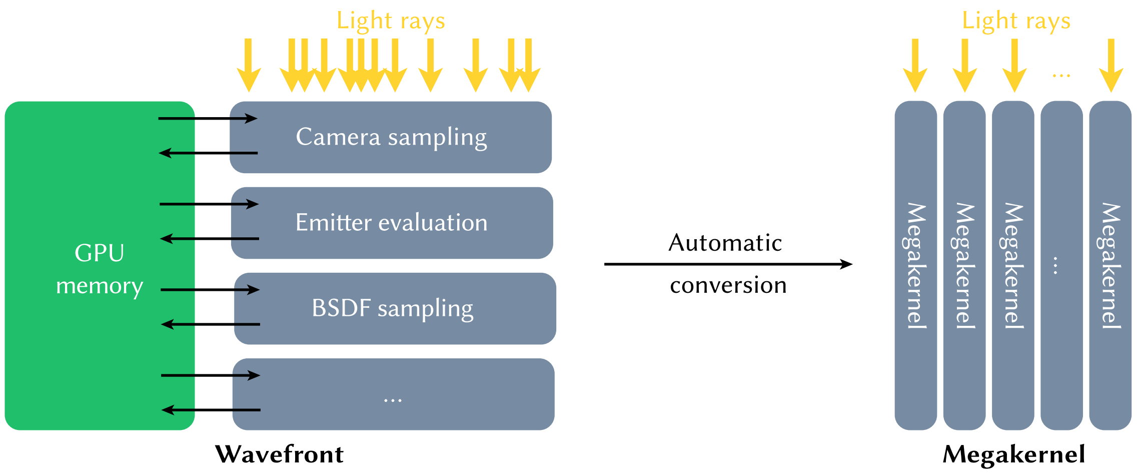


Figure 8.1: Mitsuba 2's operates in wavefront mode: relatively small kernels advance all rays in lockstep, and output the new state of the program to global memory. This allowed us to amortize the cost of maintaining AD data structures over a large number of rays. With radiative backpropagation, usage of AD can be limited to small, self-contained functions, which makes it possible to move to megakernel-style rendering. We propose a simple modification of Enoki's JIT compiler to automatically translate Mitsuba 2's components into primal and adjoint kernels that can then be attached to an OptiX megakernel.

In Chapter 4, we introduced radiative backpropagation, an adjoint method that reformulates the gradient estimation problem as a modified light transport integral. In turn, this enables a much more localized application of AD. Since the bulk of the computation graph is replaced by radiative backpropagation, the memory footprint is correspondingly much lower. Most importantly, radiative backpropagation admits a megakernel-style implementation. This chapter describes modifications of Mitsuba 2 enabling an automatic translation from wavefront to megakernel-style rendering, leading to performance improvements of up to 1000×.

8.1 Wavefront rendering in Mitsuba 2

Enoki's lazy JIT. As described in Chapter 7, Mitsuba 2's GPU targets perform arithmetic and other operations via Enoki's `CUDAArray<T>` type, which lazily fuses operations into larger kernels for later execution on the GPU. For example, line 3 of the C++ fragment:

```
1 using Float = enoki::CUDAArray<float>;
2 Float a = /* .. */, b = /* .. */;
3 Float c = a * b;
```

does not immediately perform a multiplication but rather records that a multiplication should take place when its evaluation can no longer be postponed. Note that `a`, `b`, `c` would typically have millions of entries—one per wavefront element, which typically correspond to light rays.

Certain operations create barriers that force the system to close and emit a fused kernel using NVIDIA's PTX intermediate assembly language that is then compiled to machine instructions and executed. An example of this is ray tracing via the OptiX framework, which requires concrete ray origins and directions rather than symbolic descriptions. Communication between separate kernels occurs by reading and writing large GPU-allocated arrays. These expensive memory operations, as well as their large footprint, often become the main bottlenecks of this wavefront-style rendering.

Autodiff in wavefront-style rendering. In Mitsuba 2's AD-based modes, the arithmetic types are furthermore wrapped into `DiffArray<T>`, which realizes forward- and reverse-mode differentiation on top of `T`. In the following example,

```
using Float = enoki::DiffArray<enoki::CUDAArray<float>>;
Float a = /* .. */, b = /* .. */;
Float c = a * b;
enoki::backward(c);
```

operations involving `Float` variables are recorded in a transcript, or computation graph, used for reverse-mode traversal in `enoki::backward()`. Because `DiffArray<T>` carries out its arithmetic via the underlying type `T`, additional AD-related arithmetic is also fused into PTX kernels. While the above examples were extremely simple, the same principles hold for larger system components: when the system evaluates a rough dielectric microfacet model or samples a direction from an environment map, the implementations return immediately, having recorded their operations symbolically in both AD transcript and fused PTX.

One serious problem with this approach is that the reverse-mode propagation via `enoki::backward()` must occur all the way at the end of differentiable rendering, and the transcript thus becomes very long. Furthermore, ray tracing and virtual function calls constitute two sources of frequent barriers that require flushing queued operations. As a consequence, temporaries and other variables required by `enoki::backward()` can no longer retain their symbolic form—they are evaluated and stored in GPU-resident arrays. Memory capacity and bandwidth remain limiting factors despite the periodic graph simplifications carried out by Enoki.

In the following, we propose a small modification to Enoki's JIT compiler enabling the generation of functions that are usable in a megakernel setting, and which avoids these memory-related issues.

8.2 Megakernel translation

We propose a practical and efficient way to implement radiative backpropagation. Building on top of Mitsuba 2, we use the tracing JIT compiler described above to produce both primal and adjoint shaders of individual system components (*e.g.* BSDFs, light sources, volumes, etc.) that can then be attached to scene objects in OptiX to enable efficient GPU-accelerated radiative backpropagation. In effect, our system automatically translates a wavefront-style renderer to a megakernel at runtime (Figure 8.1). This approach consumes no extra memory for differentiation apart from a single array that is used to store gradients. Section 8.3 demonstrates the superior performance and scalability of this design compared to Mitsuba 2's AD-based backend.

8.2.1 Wavefronts and megakernels

Seen from a high level, wavefront-based rendering systems propagate a set of rays in lockstep, using large memory regions to record their current state during this process. Separate steps of the algorithm (ray tracing, scattering, direct illumination sampling, etc.) are implemented as relatively small and independent kernels that parallelize over the wavefront and mutate its state in global memory.

As the name suggests, a megakernel instead consists of a single large kernel that includes all components of the original rendering algorithm. This kernel processes one light path at a time, generally using processor registers rather than global memory to record its state. Many instances of this megakernel execute concurrently and asynchronously. Wavefront and megakernel-based rendering each have distinct advantages and disadvantages—we refer to Laine et al. [168] for a thorough discussion. The recent emergence of GPUs with hardware-accelerated ray tracing functionality has tipped the scales towards the latter approach, since they are highly optimized for megakernels. In fact, NVIDIA's OptiX library [167] uses a megakernel architecture.

Note that Mitsuba 2's approach to differentiable rendering, described in Chapter 7, would not have been practical to implement as a megakernel due to the need to maintain and traverse a transcript, which involves a graph and numerous auxiliary data structures including red-black-trees and multiple hash tables. Mitsuba 2 uses wavefronts, *i.e.* vectorial AD, precisely to amortize the runtime cost of maintaining these data structures, which is impossible in a megakernel because each element is processed individually.

8.2.2 Transitioning to a megakernel

We observe that our radiative backpropagation algorithm, presented in Chapter 4, behaves mostly like a path tracer: all remaining challenges related to differentiable rendering have been pushed into the "effective emission term" Q of Equation (4.1). If this function could somehow be turned into a "shader" that requires no access to dynamic AD-related data structures, then standard tools for megakernel-based rendering (e.g. OptiX) would be applicable.

Motivated by this idea, we decided to convert Mitsuba 2 from a wavefront architecture into a megakernel, while at the same time adding adjoints of all relevant system components referenced in Q (emitters, BSDFs, participating media). If carried out manually, both are complex changes that would require a substantial redesign of the system[1].

In its normal mode of operation, the JIT compiler records a symbolic representation of each instruction until a synchronization point triggers evaluation, at which point queued computations are compiled and executed on the GPU. We instead build on the JIT compiler of Enoki to automatically perform this transformation. Our approach is based on a simple modification of this scheme: we simply interrupt the process following code generation and return a string representing the generated program (using NVIDIA's PTX intermediate representation) rather than executing it. These functions can then be attached to scene objects and will trigger computation following ray intersections (in OptiX terminology, this is called a "closest hit program"). This is illustrated

[1]We note that source transformation-based AD tools such as ADIFOR [170] or Tapenade [171] could in principle perform the latter step, but their restriction to FORTRAN / plain C fragments makes them challenging to reconcile with the highly object-oriented nature of modern renderers.

in Figure 8.2. In the end, the only manual CUDA implementation remaining is the radiative backpropagation code itself which pulls these fragments together, representing less than a thousand lines of code for the surface case. Listing 11 shows how the primal evaluation routine of a BSDF can be extracted using this simple recording mechanism.

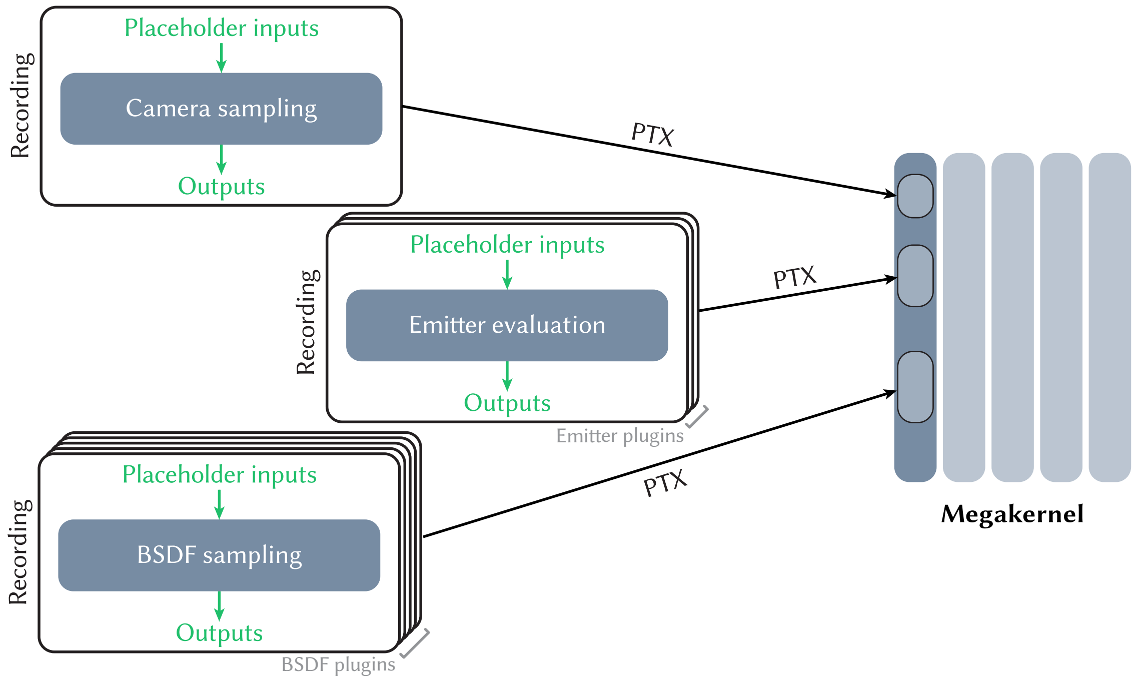


Figure 8.2: We modify Enoki's JIT compiler to extract the PTX representation of individual functions rather than appending them to a kernel and launching it on the device. The functions to be extracted are executed on placeholder inputs of size 1, so that the resulting code can be seamlessly called from an OptiX-style megakernel renderer.

Listing 12 shows how adjoints can be extracted using the same mechanism, which entails four additional steps:

1. Declaration of an additional input: the *gradient w.r.t. the output* of the primal function.

2. Declaration of differentiable parameters, causing subsequent arithmetic from *e.g.* `bsdf.eval()` to be recorded onto the AD transcript.

3. Reverse-mode traversal via `enoki::backward()`. This step is also symbolic and appends further instructions to the recording.

4. Recording of an atomic instruction (`enoki::scatter_add()`) that will accumulate the resulting gradient(s) into the vector $\boldsymbol{\delta}_{\mathbf{x}}$.

```cpp
// Declare inputs that act as placeholders
SurfaceInteraction3f si = enoki::zero<SurfaceInteraction3f>();
Vector3f             wo = enoki::zero<Vector3f>();

enoki::start_recording("bsdf_eval");

// Execute BSDF sampling routine *symbolically*
Spectrum result = bsdf.eval(si, wo);

// Declare inputs and outputs. The PTX string will then expose
// the recorded computation as a function with these arguments
enoki::set_inputs(si, wo);
enoki::set_outputs(result);
enoki::stop_recording();

// ... record other functions ...

const char *ptx_string = enoki::ptx_module();
```

Listing 11: We automatically transform Mitsuba 2 from a wavefront renderer into a megakernel-based architecture. To do so, we make a small modification to Enoki's lazy just-in-time compiler that allows it to briefly pause execution while recording all operations involving GPU arrays into a PTX instruction sequence. This sequence can subsequently be extracted and attached to scene objects. The above example extracts the evaluation routine of a BSDF.

The resulting code fragment provides an efficient sparse implementation of the δ_x += `adjoint([[ q(z) ]], gradient)` operation discussed in Section 4.3. Importantly, the machinery of reverse-mode AD is used during code generation, but is no longer needed at evaluation time.

8.3 Performance evaluation

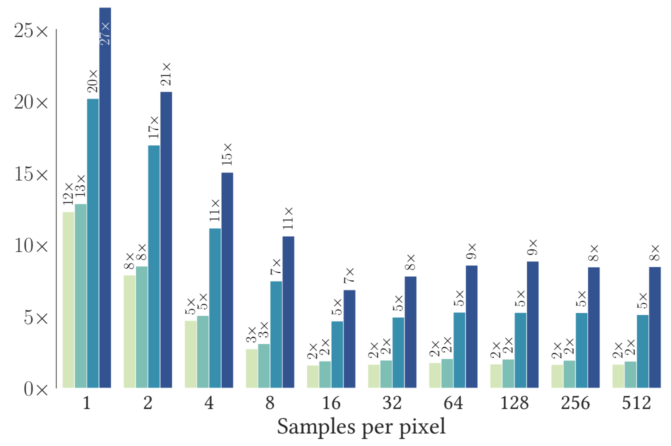
We compare radiative backpropagation's runtime, using the megakernel implementation described in this chapter, to Mitsuba 2's AD backend on four typical applications of differentiable rendering in Figure 8.3. For a given sample count, we run at least 10 iterations and measure the median runtime. We then report the relative speedup of each variant over Mitsuba 2. Ray tracing was performed by OptiX on the GPU in both cases.

We observe two performance regimes: at low sample counts, even though the full wavefront and AD transcript fit in memory, a constant performance offset arises due to

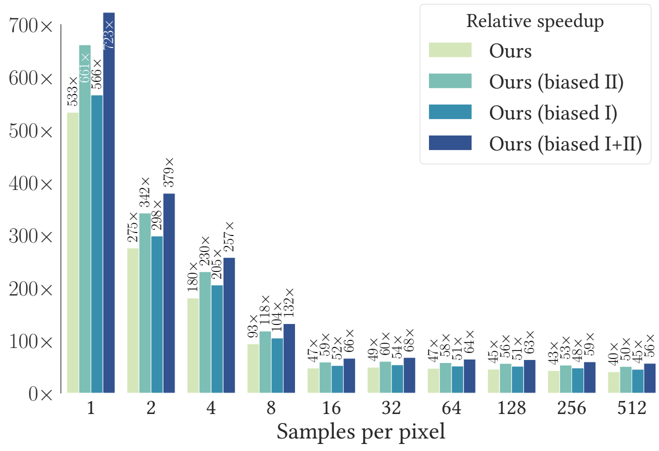
```
// Declare inputs that act as placeholders
SurfaceInteraction3f si = eknoki::zero<SurfaceInteraction3f>();
Vector3f             wo = ek::zero<Vector3f>();

// The gradient of the function's output
// is an input of the adjoint
Spectrum grad_output = ek::zero<Spectrum>();

// Reference to a BSDF parameter (e.g. roughness)
Float &param = /* .. */;

eknoki::start_recording("bsdf_eval_d");

// We want to keep track of derivatives wrt 'param'
eknoki::set_requires_gradient(param);

// Evaluate the BSDF symbolically & record a transcript
Spectrum result = bsdf.eval(si, wo);

// Backpropagate 'grad_output' to 'param'
eknoki::set_gradient(result, grad_output);
eknoki::backward<Float>();

// The derivative shader only has inputs and accumulates
// its outputs into a global array storing parameter gradients
eknoki::set_inputs(si, wo, grad_output)
eknoki::scatter_add(/* |δx| entry */, eknoki::gradient(param));

eknoki::stop_recording();
```

Listing 12: We furthermore use Enoki to differentiate components of the renderer via reverse-mode AD, recording the resulting instruction sequence symbolically. The resulting “adjoint shaders” operate without access to AD-related data structures and atomically accumulate gradients with respect to inputs into a global array.

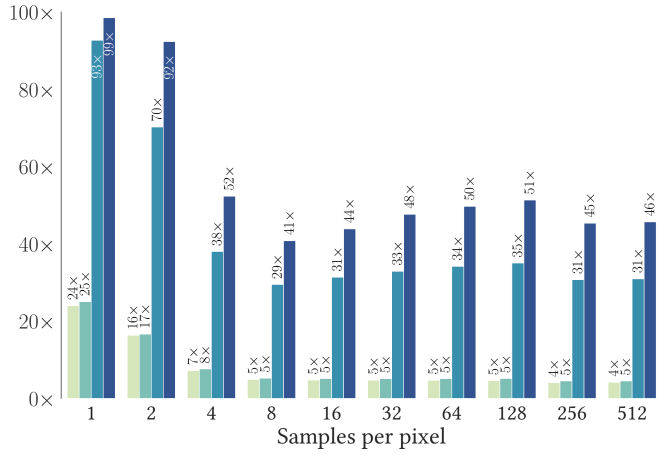
architectural characteristics described in Section 8.2.1 (wavefront versus megakernel). At higher sample counts, Mitsuba 2's AD approach quickly saturates the available GPU memory, and its computation must therefore be split into several passes. In that regime, we obtain a linear speedup of up to three orders of magnitude. As was already seen in Section 4.4.2, biased variants of our method achieve the highest speedup per iteration, while simultaneously improving convergence in most cases. These benchmarks were run on an NVIDIA 2080 Ti GPU with 12GiB of RAM.



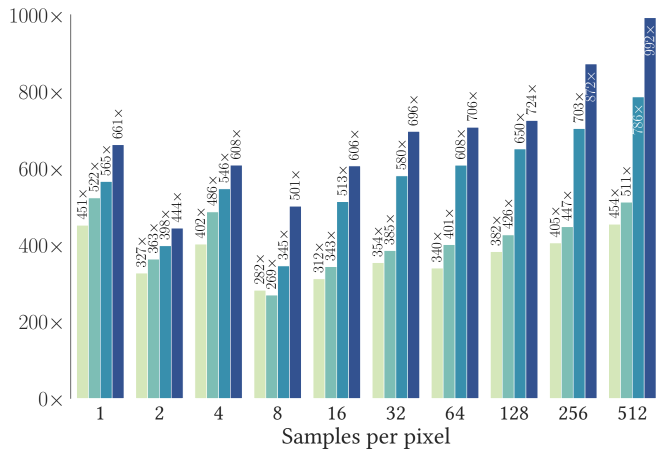
(a) Cornell box (all materials differentiable).



(b) Figure 4.1 scene, involving complex light transport.



(c) Optimization of spatially-varying albedo of a homogeneous medium



(d) Recovering the density of a smoke plume via differentiable volumetric path tracing.

Figure 8.3: We evaluate the relative performance of our method to Mitsuba 2's autodiff-based backend. For each sample count, we time at least 10 iterations of a realistic optimization. We then report the ratio of median iteration runtimes. Radiative backpropagation is up to three orders of magnitude faster.

8.4 Conclusion

The large memory consumption of AD graphs in Mitsuba 2 proved to be its largest limiting factor when going beyond low-resolution or low-complexity inverse rendering applications. The radiative backpropagation algorithm replaces most of the AD graph with an algorithm akin to path tracing. Usage of automatic differentiation is pushed to smaller, self-contained functions such as BSDF and emitter evaluation. In order to fully exploit this new flexibility without having to re-implement all system components, we modified Enoki's JIT compiler to extract both primal and adjoint kernels from the relevant Mitsuba 2 plugins at runtime.

This megakernel-style radiative backpropagation implementation achieves speedups of up to three orders of magnitude when compared to Mitsuba 2's AD backend, at a fraction of the GPU memory consumption. Furthermore, we preserved the flexibility and extensibility of Mitsuba 2: new components can be added as plugins, and the derivatives will be automatically computed locally by AD just as before.

Limitations. There are several limitations to our approach: when components of the renderer execute loops, we can only extract a correct instruction trace if the length of this loop is known or can be bounded. Fortunately, this is the case for standard loop constructs found in renderers: for instance, the number of steps needed for hierarchical sample warping or bisection of a discrete CDF is related to the resolution of the underlying distribution and thus exactly known, and root-finding techniques such as Newton iterations can be bounded with a conservative iteration count (see Section 7.3.4).

Since we effectively unroll all computations during the JIT recording step, our approach is suboptimal for long-running loops. In fact, the heterogeneous volume sampling code of Mitsuba 2, which relies on a ray marching loop, generated an unrolled kernel of more than 10MiB of PTX source. When added to the OptiX megakernel, compilation exhausted the system memory and eventually led to a crash. As a workaround, we selectively re-implemented the relevant loop in CUDA while still relying on automatic extraction of the remaining heterogeneous medium-related functionality (*e.g.* trilinear interpolation of medium parameters like σ_t).

Like normal rendering, radiative backpropagation is "embarrassingly parallel", and can thus be parallelized over many cores of CPUs/GPUs or even multiple machines. One potential issue in this context is that adjoints of shaders atomically accumulate gradients into a set of global variables, which can lead to memory contention. Allocating dedicated memory regions for each core that are merged at the end of the adjoint phase could potentially alleviate such resource conflicts. In practice, we found atomic operations to be surprisingly fast (at least on the NVIDIA GPU used for testing), and contention to be limited in the case of textured or volumetric parameters, since the accumulation is spread over many variables.

Impact. Our approach to megakernel generation in Enoki served as a starting point for the Dr.Jit [80] library. Dr.Jit fully automates and generalizes the kernel extraction technique presented in this chapter, and adds support for key features such as symbolic execution of loops. This effectively removes the limitation mentioned above. Dr.Jit is the foundation for Mitsuba 3 [79], which prominently features the radiative backpropagation algorithm and megakernel execution in order to address all AD-related performance limitations found in Mitsuba 2.

In future work, the performance characteristics of the new system should be thoroughly examined to identify the new bottlenecks. For example, we may find register spilling to be an issue in larger kernels with many intermediate results. Dr.Jit allows users to manually introduce kernel boundaries, but automatically determining the optimal kernel size is highly challenging, as it likely changes with the register count and memory bandwidth of the underlying hardware.

Part III

Applications

Over the course of Parts I and II, we have built algorithms and systems dedicated to the *correct* and *efficient* estimation of gradients with respect to scene parameters in the presence of global illumination, complex material models, and participating media.

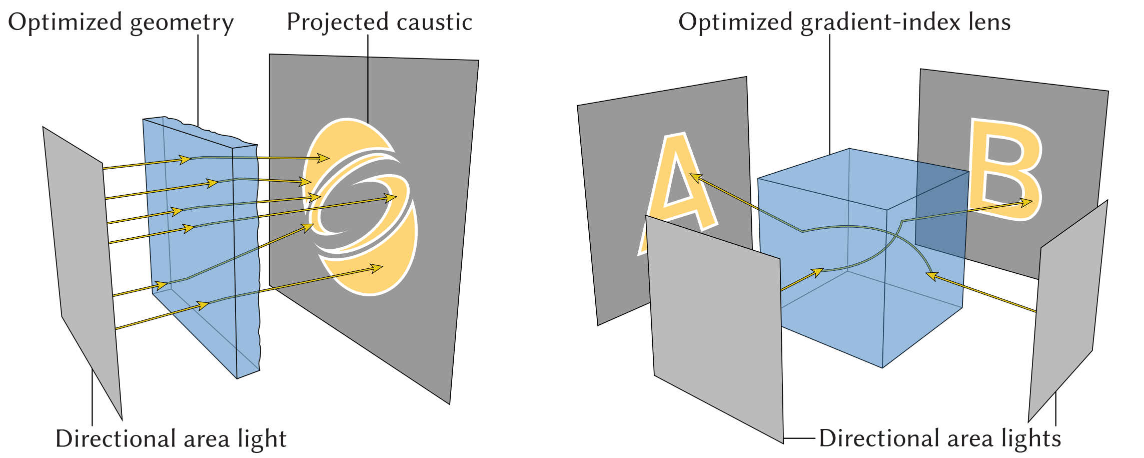
This capability unlocks a wide range of applications, from object & scene reconstruction, to augmenting artist workflows for asset creation, to broader scientific applications such as microscopy and atmospheric science. In the following chapters, we showcase three such applications.

Chapter 9 demonstrates how AD-based differentiable rendering can be used to optimize the surface of a slab of glass so that it projects a desired caustic pattern or picture as light refracts through it. In Chapter 10, we reconstruct the spatially-varying density and albedo of complex participating media. Finally, we recover high-resolution materials and emission characteristics of indoor scenes from real-life photographs in Chapter 11.

9 | Caustic optimization

Relevant background: Section 2.6 and Chapter 7.

Despite the limitations discussed in Chapter 3, automatic differentiation remains a highly useful tool for differentiable rendering—at least in low-complexity scenes where the memory usage does not become prohibitively high. Mitsuba 2, presented in Chapter 7, provides end-to-end automatic differentiation support, which greatly facilitates experimentation on gradient-based applications. In this chapter, we present two methods for computational caustics that optimize either the geometry of a glass slab or the index of refraction of a gradient-index lens so that they project a desired image onto a target surface. The corresponding experimental configurations are shown in Figure 9.1.



(a) Optimizing surface displacement

(b) Optimizing gradient-index optics

Figure 9.1: We showcase two material design and reconstruction applications that optimize **(a)** a refractive height field focusing collimated illumination into a desired image on a target surface; and **(b)** a cube with spatially-varying index of refraction that propagates light along curved rays, encoding two separate images for illumination arriving from perpendicular directions. *Illustration by Tizian Zeltner, reproduced from [14].*

9.1 Surface displacements

In this first application, the surface of a slab of glass is optimized as a high-resolution height field, modifying it until the light from a collimated light source passing through it is focused into the desired caustic pattern. A fixed camera observes the pattern projected onto a diffuse plane and compares it to a goal image.

Related work. A similar problem setting has been studied by Papas et al. [172], who used a decomposition of Gaussian kernels and Yue et al. [173], who solve a sequence of Poisson problems to construct a smooth height field. Schwartzburg et al. [174] used a tailored optimization formulation based on optimal transport. In contrast, our method simply entails implementing the forward simulation in Mitsuba 2, and optimizing the relevant parameters using the computed gradients.

Rendering algorithm. We render caustics using a unidirectional *light* tracer (also called *particle tracer*) rather than a path tracer. Indeed, we assume a collimated light source, which has a delta (or very narrow) directional distribution. Using path tracing, paths started from the camera and being refracted by the optimized lens surface would reach the light source at the right angle with a vanishingly small probability. Next-event estimation, which samples a refraction direction according to the emitter's distribution, would also fail as the glass surface's BSDF is equally narrow.

Using light tracing, paths are started from the light source, traced in the collimated direction, refracted by the lens, and hit the (large) diffuse plane with high probability. Finally, using next-event estimation, a connection is made from the diffuse plane to the camera, which always succeeds. Note that this method is unbiased, and converges well enough that we do not need to resort to approximations such as photon mapping.

Differentiable image formation. For AD-based optimization, using a differentiable image reconstruction filter is crucial to capture the relationship between the lens' geometry and the final brightness and position of the various parts of a caustic. In particular, a discontinuous "rectangle" or "box" filter will *not* yield any gradient w.r.t. the lens' surface heightfield. The relationship between the heightfield and each pixel of the rendered image is as follows:

1. At the intersection point between the light ray and the lens surface, the surface's normal direction is computed as a function of the heightfield displacement values.
2. The refracted ray direction is computed as a function of the surface normal.
3. The refracted ray intersects with the diffuse receptor plane, participating in the creation of a caustic pattern. The local intersection point with the plane is computed as a function of the ray direction. The amount of reflected radiance is also affected by the incident angle via the foreshortening term ($\cos\theta$).

4. A path segment is sampled explicitly to connect with the sensor. The specific intersection point between the ray and the image plane is computed as a function of the position of the ray origin on the diffuse plane.

5. Finally, the path's radiance is added to the image, weighted by the reconstruction filter. Here, the weights from a constant box reconstruction filter would be constant w.r.t. the intersection point on an image plane. This is why it is crucial to use a differentiable, non-unit filter such as a Gaussian kernel to maintain the continuous relationship between heightfield and final image values.

Results. Figure 9.2 (a-c) shows three results and intermediate optimization states for displacement-based caustics. While these results do not match the quality of a purpose-built system in terms of contrast and precision, its ease of use is appealing: the only requirement is a suitable forward simulation implemented in our AD-based system, which can then be optimized using a variant of gradient descent. The method's generality makes it immediately applicable to broader settings. For instance, in Figure 9.2 (c), the optimization generates geometry that blends primary colors from multiple light sources in the right proportions to create a color image.

Experimental conditions. We optimize the lens heightfield using the SGDm optimizer over 350 iterations. At each iteration, we render a 512×512 image with 120 samples per pixel and compare it to the goal image via a scale-independent L_2 loss function. Gradients are estimated with 20 samples per pixel. The heightfield is initialized to 0 (flat lens surface), with a resolution increasing from 32×32 to 1024×1024 over the course of the optimization.

Limitations. Note that discontinuities such as the edges of the receptor plane or the camera's field of view, or due to total internal reflection, were not accounted for in our experiment. While we believe these do not play a significant role in the end result, a thorough evaluation should be conducted in future work.

We also attempted optimizing both sides of the glass slab but found the resulting caustics to have worse contrast and detail. The issues persisted in various optimization schemes, such as optimizing the two sides in sequence or assigning a lower resolution to one of them. Finally, we did not attempt to control stray light: paths that are redirected out of the field of view during optimization will no longer produce any gradient and therefore remain fixed.

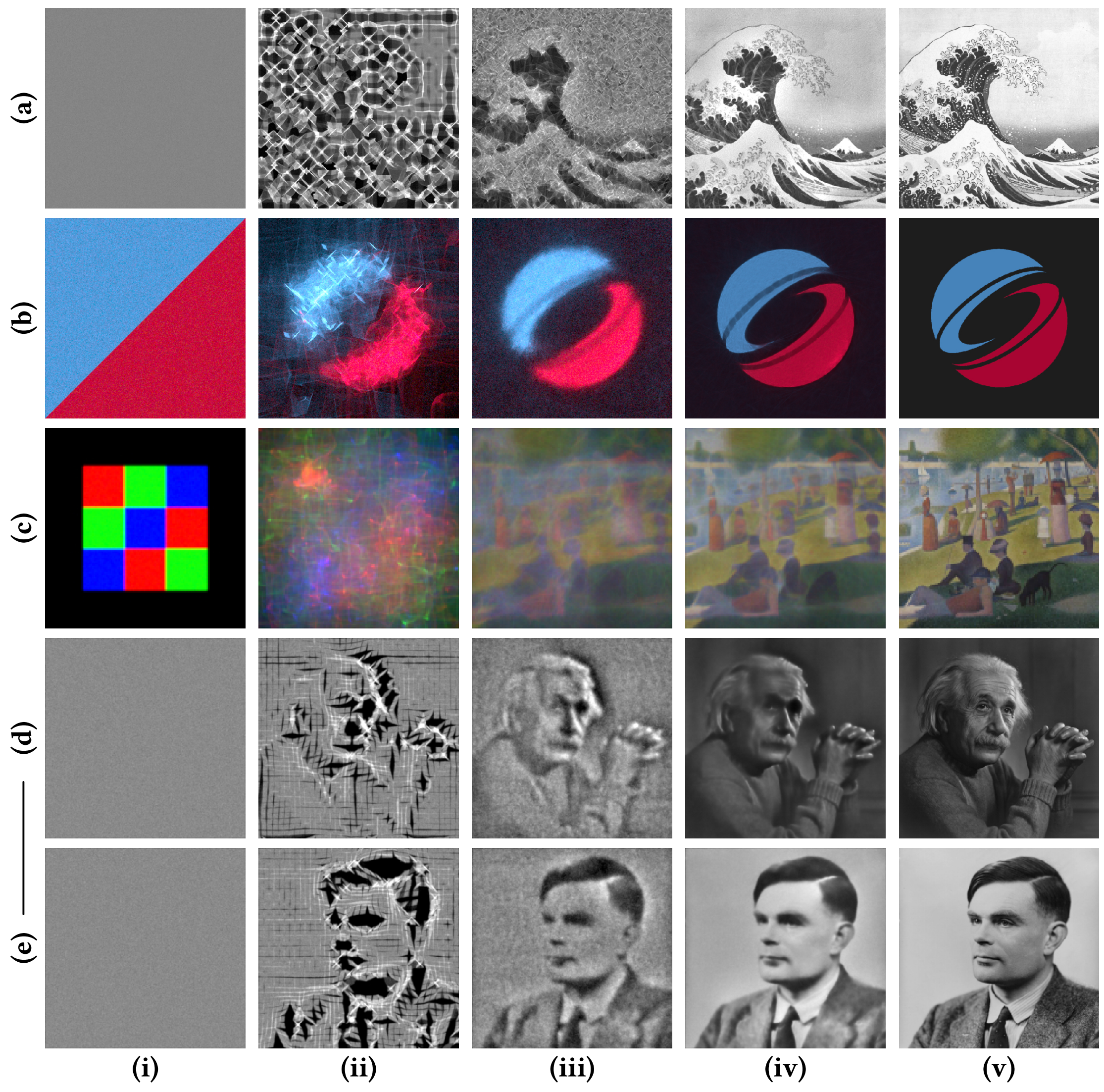


Figure 9.2: We optimize **(a–c)** refractive height fields to focus collimated light into a desired image (Figure 9.1a), and **(d–e)** the spatially-varying index of refraction of a *single* gradient-index lens to project different images when illuminated from two incident directions (Figure 9.1b). **(i)** starting from a uniform solution that simply refracts light through, **(ii–iii)** the optimization quickly approximates the main features of the target image **(v)**. We render the final state **(iv)** after matching the emitters' intensity, which was not part of the optimization. The optimization targets used are the woodblock print "The Great Wave off Kanagawa" by Hokusai, the ACM SIGGRAPH logo, the painting "A Sunday Afternoon on the Island of La Grande Jatte" by Georges Seurat, a 1948 photographic portrait of Albert Einstein by Yousuf Karsh, and a 1951 photographic portrait of Alan Turing by Elliot & Fry Studio.

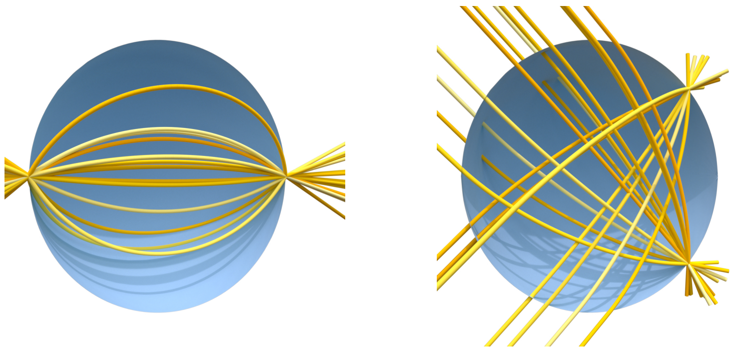


Figure 9.3: Example gradient-index lenses: the Maxwell fish-eye lens (left) images a point onto an antipodal point, while the Luneburg lens (right) collimates incident light.

9.2 Gradient-index optics

Our second application concerns so-called *gradient-index optics* or "GRIN" lenses.

GRIN lenses. In materials with a varying index of refraction, light travels along curved rays according to the Eikonal equation. Expressed as a second-order ODE [175], it relates the change in position to the gradient of the refractive index:

$$\frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2} = n(\mathbf{x})\,\nabla n(\mathbf{x}). \tag{9.1}$$

Gradient-index optics exploit this effect to create lenses that lack the typical aberrations of spherical lens elements. For instance, the (mostly theoretical) Maxwell fish-eye lens with a radially symmetric index of refraction ($\eta(r) = {}^{1}/_{1+r^2}$) or the Luneburg lens ($\eta(r) = \sqrt{2 - r^2}$) image a point onto an antipodal point or collimate it, respectively. Their behavior is illustrated in Figure 9.3.

Algorithm. Integrating a differentiable ODE solving step in Mitsuba 2 is easy—we reproduce the central solver loop used to create this result in Listing 13. Lines 6, 7, and 15 are the Leapfrog discretization of the Eikonal Equation (9.1), where `fmadd` denotes a fused multiply-addition operation. Lines 8–11 keep track of which lanes have exited, saving the final state when rays leave the material. Line 12 is the stopping condition, written in such a way that the potentially costly horizontal reduction `none(active)` is skipped as long as it is likely that at least one ray remains inside.

Results. Fabrication of materials with a varying index of refraction is an active area of research [176] that could one day reduce the cost of current aspherical optics. Here, our

```
Point3f p_out;
Vector3f v_out;
Mask active = true;
for (size_t i = 0;; ++i) {
    auto [ior, ior_grad] = evaluate_ior(p, active);
    Vector3f v_half = fmadd(.5f * step_size * ior, ior_grad, v);
    Point3f p_next = fmadd(step_size, v_half, p);
    Mask escaped = active && !is_inside(p_next);
    active &= !escaped;
    p_out[escaped] = p;
    v_out[escaped] = v;
    if (i >= 2.f / step_size && none(active))
        break;
    p = p_next;
    v = fmadd(.5f * step_size * ior, ior_grad, v_half);
}
```

Listing 13: Enoki-based Eikonal equation solver with a "leapfrog" discretization.

differentiable renderer already provides a helpful tool for optimizing the properties of such a material based on a user-specified objective function. Figure 9.2 (d–e) shows two caustics that are *simultaneously* projected by a single gradient-index cube illuminated from two perpendicular directions. Note that the discrete appearance of intermediate optimization steps is due to the trilinear interpolation of refraction values, which causes piecewise constant gradients in the ODE in Equation 9.1.

Experimental conditions. We optimize the spatially-varying index of refraction of a glass cube using the SGDm optimizer over 500 iterations. At each iteration, we render two 256×256 images, one for each receptor plane, with 160 samples per pixel and compare them to the two goal image via a scale-independent L_2 loss function. The overall optimization objective is simply the sum of the two. Gradients are estimated with 16 samples per pixel. The heightfield is initialized to 1.5 (corresponding to standard glass), with a resolution increasing from $5 \times 5 \times 5$ to $160 \times 160 \times 160$ over the course of the optimization.

9.3 Conclusion

With minimal implementation effort, we obtain convincing results in two caustic design applications, where existing methods typically require manual derivations and purpose-built algorithms. Thanks to automatic differentiation, we can easily generalize our implementation from relatively straightforward heightfield-based grayscale caustics; to color caustics; to a GRIN lens projecting two different images at once.

Limitations. Unfortunately, the limitations of AD-based inverse rendering discussed in Chapter 3 still apply. The rendering quality (image resolution and sample count), as well as the overall runtime performance, were limited by the AD system's memory usage. Whenever possible, an adjoint method such as radiative backpropagation (Chapter 4) should be preferred. In fact, Teh et al. [177] derived an adjoint method for differentiable GRIN lens rendering, successfully reproducing the experiment of Section 9.2. They also showcased applications to multifocal displays, optic fiber design, and fuel injection scenarios.

Future work. Our experimental setup is idealized and does not account for important real-life considerations such as manufacturability of the resulting lens, calibration of the camera and light source, measurement error, the sensitivity of the lens to the focus distance, the impact of stray light, etc.

Even in ideal conditions, the optimization problem remains highly non-convex. Moreover, first-order gradients and a pixelwise error function are not sufficient to handle cases where bright regions of the caustic must be moved over large distances in image space—the gradients are simply not informative enough.

Finally, broader applications such as automatic imaging lens design [129] and glare minimization for architecture should be investigated.

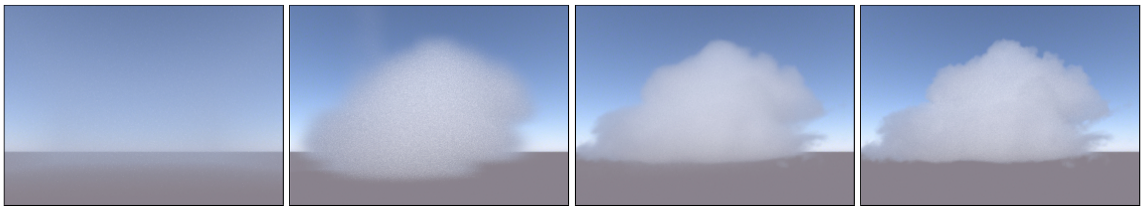
10 | Inverse volume rendering

Relevant background: Section 2.2.3 and Chapter 5.

Participating media are known to be challenging to render—especially when they have spatially varying densities and are highly scattering. Intuitively, this is due to the many internal scattering events that need to be simulated in order to account for all of the radiance propagating within the medium. *Inverse* rendering of such volumes, then, is all the more difficult.

The algorithmic contributions of Chapters 4 and 5, as well as the systems work of Chapters 7 and 8, enable us to tackle the inverse volume rendering problem. The combination of these tools makes it possible to support the full physically based light transport simulation with scattering heterogeneous media in an efficient and unbiased way. As an example, recovering the scattering coefficient of a dense, high-albedo cloud *requires* such this accurate, physically based simulation. The reconstruction shown in Figure 10.1 hints at applications of differentiable rendering in the fields of climate science.

In this chapter, we discuss remaining challenges and present reconstruction results using the unbiased estimator of Chapter 5.



(a) Iteration 0 (b) Iteration 1000 (c) Iteration 12000 (d) Reference

Figure 10.1: We recover the spatially-varying density of a simplified version of the Walt Disney Animation Studios cloud. The scene is lit by a realistic environment map with a sun $\sim 50000\times$ brighter than the surrounding sky, which makes even primal rendering particularly challenging.

10.1 Combating local minima

In Chapter 5, we have derived an estimator tailored to the estimation of gradients with respect to heterogeneous medium parameters, eliminating the bias and variance found in existing approaches. However, our experience shows that regardless of the estimator used and the quality of the gradient estimates, inverse rendering of scattering media using gradient-based optimization may not always converge to a global minimum. We propose leveraging nonphysical emissive volumes, whose convergence is better behaved, to *bootstrap* the optimization of our scattering media.

Indeed, the problem can be alleviated by first training a simplified model with known favorable convergence properties: a purely emissive and absorbing volume, similar to concurrent work on voxel-based radiance fields [119] and inspired by neural radiance fields (NeRF) [118]. We use the resulting densities to bootstrap our scattering volume optimization, optimizing for the unknown scattering albedos and refining the density values. Our experiments show empirically that this procedure results in significantly more accurate reconstruction; see for example Figure 10.2.

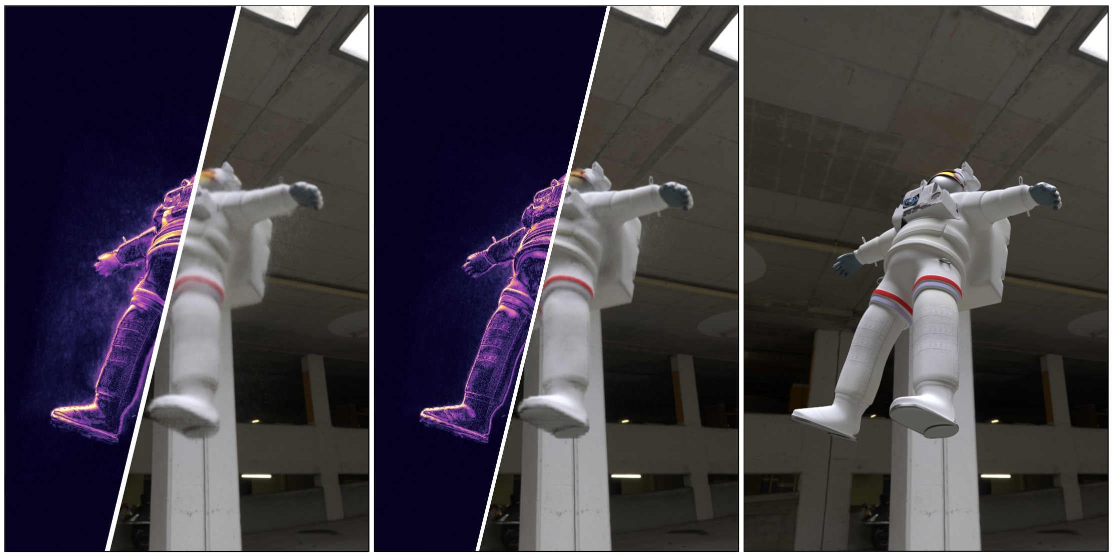

(a) Ours (constant init.) (b) Ours (emissive init.) (c) Reference

Figure 10.2: **(a)** Using the differential ratio tracking technique of Chapter 5 significantly reduces gradient variance, leading to a good reconstruction. **(b)** We further propose to initialize the scattering volume optimization from the result of a nonphysical emissive volume optimization in the style of NeRF [118]. This helps overcome local minima, greatly improving the sharpness of the final model.

10.1.1 Source of local minima

Given the complexity of the reconstruction problem and the presence of many ambiguities, it is understandable that the loss landscape may include many local minima. For an intuitive example, consider the following situation: at a given stage of the optimization, the objective function indicates that the brightness of a given pixel should be increased to better match the reference image. At each point along the paths contributing to this pixel, increasing local density increases in-scattered radiance at that point, but simultaneously reduces transmittance and therefore attenuates contribution from interactions

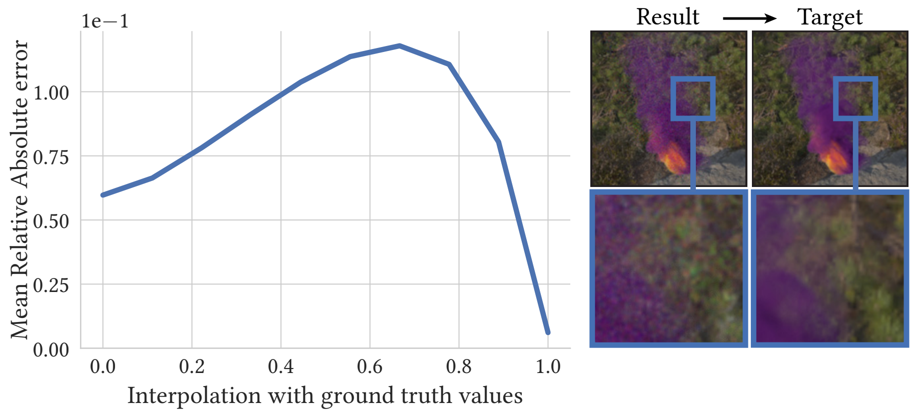


Figure 10.3: Regardless of gradient quality, inverse rendering is prone to difficult local minima. We compute the value of the re-rendering loss as we interpolate from the result of an optimization (top left image) to the ground truth value (top right image). While this is not an exhaustive exploration of the loss landscape (the volumetric model comprises 16+ million parameters), it suggests the presence of a local minimum.

further down the path. This delicate balance is captured by the terms of opposite signs in the expression of the gradients (Equation (5.3) and Equation (5.4)):

$$\begin{aligned}\partial_\theta L_i(\mathbf{x}) = &\int_0^{t_s} T(t)\; \partial_\theta\big[\sigma_t(t)\;\alpha(t)\big] L_s(t)\,\mathrm{d}t \\ &- \int_0^{t_s} T(t)\;\sigma_t(t)\;\alpha(t)\;\left[\int_0^{t} \partial_\theta\sigma_t(t')\,\mathrm{d}t'\right] L_s(t)\,\mathrm{d}t\;\;\cdots \end{aligned} \tag{10.1}$$

In Figure 10.3, we plot the evolution of the objective function as we interpolate from a converged optimization result to the ground truth solution. While this is not an exhaustive exploration of the local neighborhood (impractical due to the millions of medium parameters), it suggests the presence of a local minimum.

10.1.2 Emissive volume initialization

On the other hand, nonphysical emissive volume representations such as the one employed by NeRF [118] and subsequent works seem to converge to accurate solutions without fault[1]. In our experience, this remains true when replacing the neural network with a grid of interpolated values (similar observations were made by others [119, 178]) and, in our simple scenes, even when omitting directionally-dependent emission.

[1]Here, by “accurate” we mean low training error in image space. The correspondence of the reconstructed volume to its true shape (test error) additionally depends on a sufficiently large number of training views to constrain the problem sufficiently.

Relying on this property, we propose the following simple two steps to optimize scattering volume parameters. First, reconstruct a nonphysical emissive volume density σ_t and emission L_e. This is expected to converge fast both in terms of iteration count and runtime: the maximum path depth is one (no scattering) and variance is minimal. Second, use the recovered density σ_t to initialize the optimization of the physical medium parameters σ_t and α. The density σ_t will typically only require small adjustments—in particular when low-order scattering is the dominant source of color—because of the close resemblance between L_s and L_e in the radiative transfer equation (5.1). The albedo α, on the other hand, is easy to recover because it linearly relates to pixel color; it is not affected by the opposing gradients in Equation (10.1) that govern occlusion. We have experimented with different ways of converting the previously optimized emission L_e into initial albedo values to bootstrap the second optimization, but did not find it to perform better than initializing albedo from a constant value, *e.g.* 0.6. All initializations led to rapid convergence.

We found this two-step optimization to be particularly helpful for the inverse rendering of dense objects: see for example the increased sharpness in Figure 10.2b.

Alternative initializations. One may consider initializing the density σ_t in the second step from a number of other shape estimation techniques such as silhouette carving [179], multi-view stereo [180], or COLMAP [181]. Unfortunately, many of these methods are not designed for participating media or semitransparent objects. Even when reconstructing just solid objects, the precise density values σ_t to extract from hard surfaces are non-obvious. By bootstrapping from an emissive *volume*, despite it being nonphysical, we directly obtain an interpretable σ_t value. Moreover, emissive volume optimization is easily implemented as a special case of scattering volume optimization. Orthogonally, reparametrizations leveraging similarity relations can be exploited to improve convergence [103]. For surface reconstructions, VolSDF [182] could be substituted to the emissive volume model for a higher fidelity initialization, at the cost of implementation complexity.

10.1.3 Inverse volume rendering

Finally, we apply the combination of our differential ratio tracking algorithm (Chapter 5), emissive volume-based initialization scheme (Section 10.1.2) and just-in-time compiled megakernel inverse renderer (Chapter 8) to an inverse volume rendering application.

Given reference images and initial values for the medium properties of interest, we use gradient-based optimization to minimize a re-rendering objective function. Note that our work focuses on the effective estimation of gradients: assembling a fully robust reconstruction pipeline using real-world, imperfect data has its own set of challenges (see Chapter 11). Therefore, we use synthetic scenes with known camera parameters, illumination conditions, and isotropic phase function. Since we target non-emissive objects, we only optimize for L_e in our initial stage of fitting a nonphysical, emissive volume. In the second stage of optimization, we set medium emission L_e to zero and require all color to originate from lighting and the reconstructed scattering albedo α. Consistently resolving the additional ambiguities and complexity brought by a real-world capture setup is an important direction for future work, most likely involving domain-specific assumptions and inductive biases. An interactive viewer showcasing the inverse rendering results in this chapter is accessible at:

`https://rgl.epfl.ch/publications/NimierDavid2022Unbiased`

Experimental conditions. We implemented a differentiable volumetric path tracer with path replay backpropagation [21] in Mitsuba 3. Within the same codebase, we compare the existing free-flight sampling-based gradient estimator with our differential ratio tracking estimator (Chapter 5). As was noted in Section 5.5.2, the choice of optimizer has a large impact on the reconstruction quality in the presence of high-variance gradients. We run the optimizations using both stochastic gradient descent with momentum (SGDm) and Adam [150].

Unbiased global illumination is simulated using a path length of up to 64 vertices (although there is no particular limit, as all evaluated methods have linear time complexity in path length). The scenes are lit by realistic indoor and outdoor high dynamic range light probes.

The optimization aims to minimize the L_1 difference to 64 reference images, with camera positions sampled uniformly in a circle around the object at random altitudes. All optimizations were run for 6000 iterations. In each iteration, a batch of 32768 pixels is sampled uniformly from the set of all pixels from all reference images. Primary rays are then sampled from within the footprint of selected pixels.

In step (I) of path replay backpropagation (Section 5.2.2), the primal value is estimated with 1024 samples per pixel, for a total of 33.5 million rays. The high-quality primal estimate is used to compute the objective function. Steps (II) and (III) then use a second, uncorrelated set of primary rays with 16 samples per pixel to estimate gradients.

In each experiment, the same learning rate is used for all estimators. This value must be adapted to the specific scene's physical scale and any factor applied to the medium density. Intuitively, a scalar factor applied to the σ_t values would be reflected in the gradients, resulting in a different effective step size. In practice, we set the learning rate to values in the ranges $5 \cdot 10^{-2}$ to $5 \cdot 10^{1}$ for SGDm and 10^{-3} to 10^{-2} for Adam. It is automatically set twice as high for the albedo parameters.

Results are shown at equal iteration counts across methods, as the runtime depends not only on the chosen method but also on the state of the reconstruction itself[2]. Equal-time results would therefore be difficult to interpret.

Several commonly used techniques further improve the quality of the results: the optimization starts with 16^3 fewer parameters than the desired resolution. The grid resolution is then doubled four times during the optimization (coarse-to-fine). When using SGDm, the learning rate is multiplied by 4 at each upsampling step. Finally, the learning rate is halved six times over the last quarter of the optimization.

Inverse rendering results. We showcase equal-iterations inverse rendering results on three solid objects (Figure 10.4) and two complex heterogeneous participating media (Figure 10.6). Quantitative results are shown in Figure 10.5 and Figure 10.7 respectively. The free-flight sampling-based gradient estimator fails to converge on all scenes when used together with SGDm **(a)**. Using the Adam optimizer, high gradient variance manifests instead as a persistent thin "haze" around the reconstructed volumes **(b)**. These artifacts are visible in the included color-mapped density slices and are best appreciated by flipping back and forth between images; please see the interactive viewer linked above. Defensive sampling **(c)**, while eliminating the larger gradient outliers, does not sufficiently reduce overall variance and therefore exposes the same issues in the reconstructions.

Our proposed gradient estimator and initialization scheme achieve the best reconstructions **(d)**. The lowered gradient variance helps eliminate the haze-like artifacts while the emissive initialization helps converge to sharper results (see also Figure 10.2b). Note that the reproduction of hard surfaces with volumes could be improved by *e.g.* incorporating the non-exponential transmittance model of Vicini et al. [183].

Relighting. A great advantage of using a fully physically based and unbiased inverse rendering method is that it yields inherently editable and relightable media. In Figure 10.8, we relight the reconstructed medium using an environment that was not seen during optimization. As expected, the NeRF-style nonphysical emissive volume appears

[2]Denser media lead to more internal interactions, which increases overall runtime.

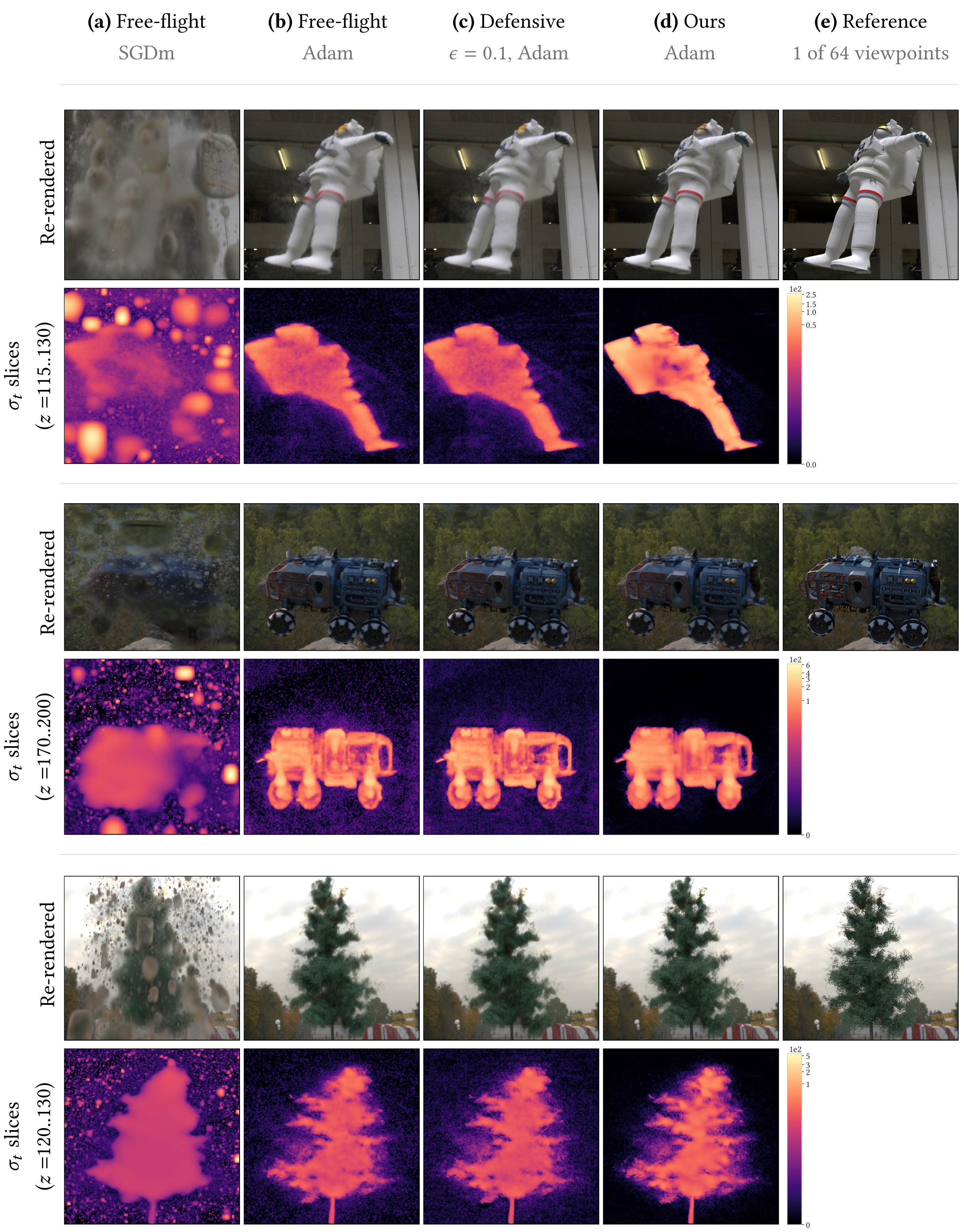
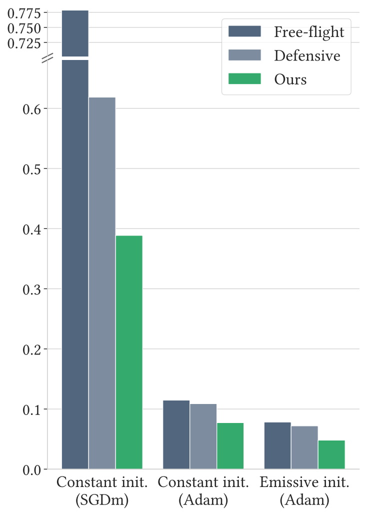
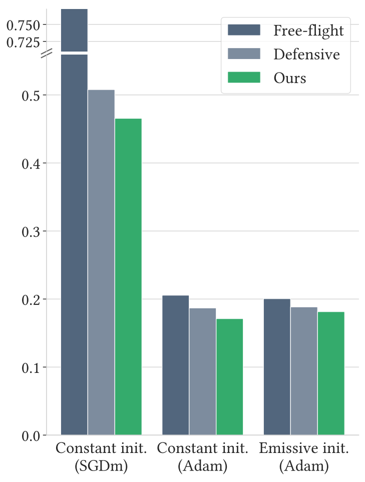


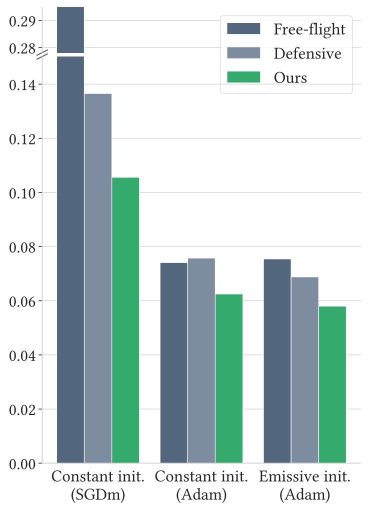
Figure 10.4: Our method enables high-quality inverse rendering of complex objects under realistic illumination. We compare inverse rendering results from different combinations of optimizer and gradient estimators. **(a)** The high-variance gradients produced by the free-flight sampling-based estimator prevent any convergence when optimizing with stochastic gradient descent with momentum. *Continued in Figure 10.5.*



(f) Astronaut scene



(g) Rover scene



(h) Tree scene

Figure 10.5: *Continued from Figure 10.4.* **(b)** The Adam optimizer's per-parameter adaptive step size significantly reduces the impact of gradient outliers. Nevertheless, persistent haze-like artifacts remain present at the end of optimization. They are also visible in the false-color density slice visualizations. **(c)** Defensive sampling (Section 5.3.2) prevents large gradient outliers but otherwise introduces additional variance, resulting in similar artifacts. **(d)** Our method uses a novel sampling scheme dedicated to the adjoint to eliminate most of the gradient variance. Combined with our proposed emissive volume initialization (Section 10.1), we achieve sharper reconstructions and eliminate haze artifacts. **(f-h)** Re-rendering loss values for the three scenes of Figure 10.4. We report the mean absolute percentage error computed on all 64 reference images after optimization.

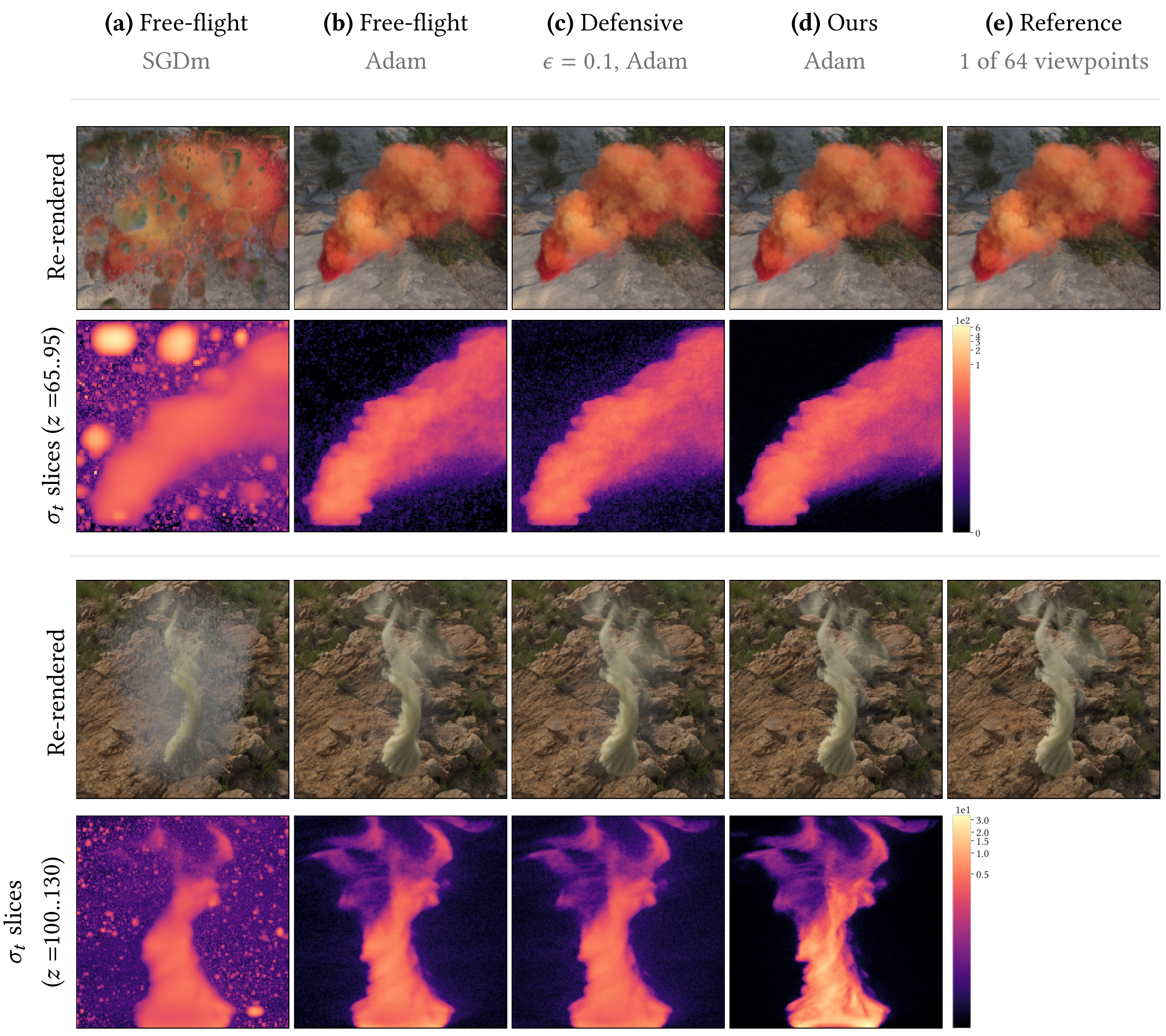
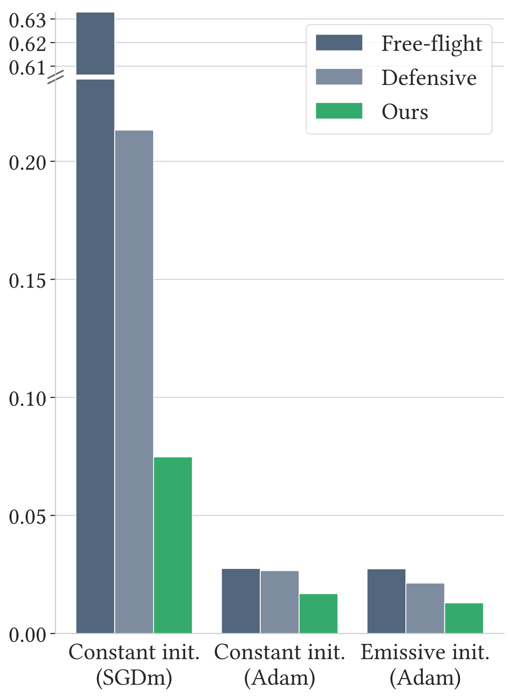
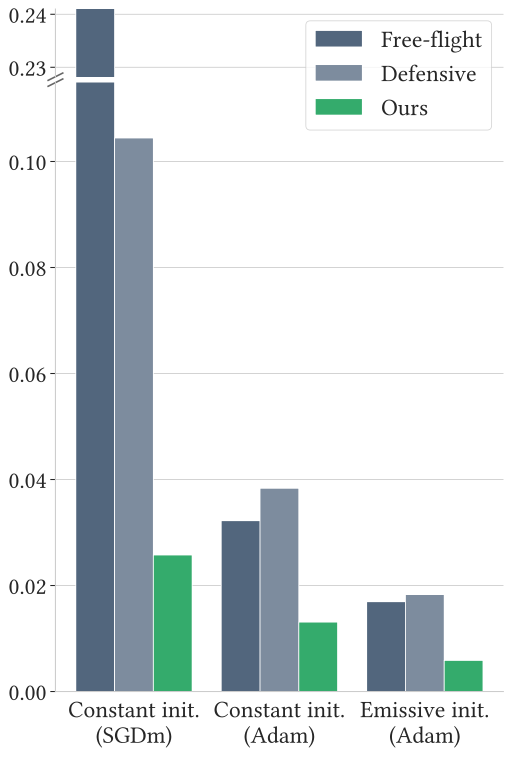


Figure 10.6: Our method enables high-quality inverse rendering of heterogeneous scattering and absorbing media under realistic illumination. *Continued in Figure 10.7.*



(f) Janga smoke scene



(g) Dust devil scene

Figure 10.7: *Continued from Figure 10.7.* We report the mean absolute percentage error computed on all 64 reference images after optimization for the two volumetric scenes of Figure 10.6.

identical regardless of the surrounding illumination, which is incorrect[3]. On the other hand, ours reacts appropriately to the new lighting conditions. Note that robustly disentangling lighting and reflectance for fully accurate relighting is challenging in its own right and likely requires observations under different lighting conditions.

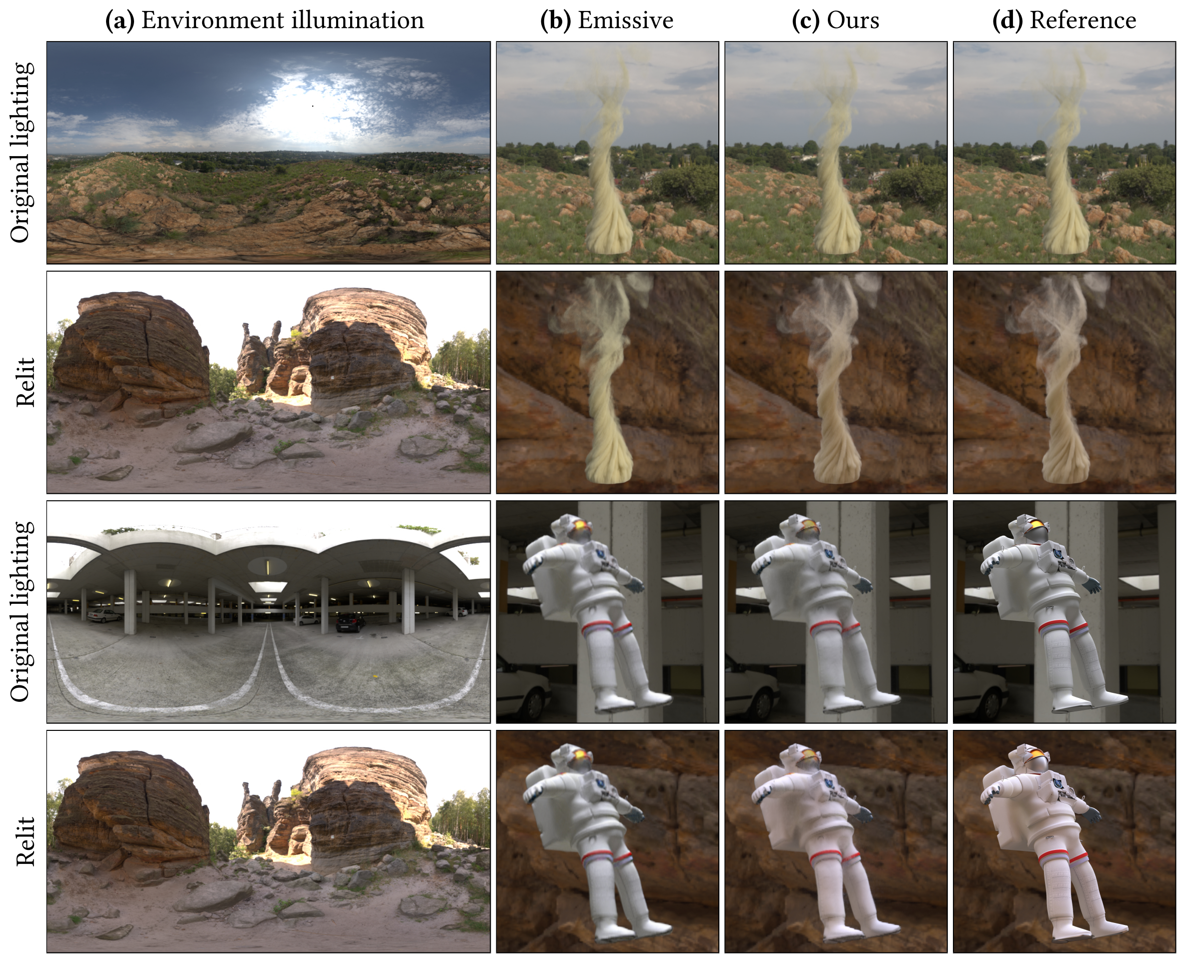


Figure 10.8: Nonphysical emissive volume models such as NeRFs [185] and Plenoxels [119] faithfully reproduce the target object, but also "bake" the original lighting **(a)** into the model parameters **(b)**. Using a fully physically based volumetric model and inverse rendering pipeline, we obtain high-resolution density and albedo medium parameters. These parameters have concrete physical meaning and are inherently editable and relightable. When re-rendered in previously unseen lighting conditions, our optimized models **(c)** react correctly to the new illumination. Note that we did not attempt to model more complex surface reflectance models (*e.g.* specular reflections), which are orthogonal to our method.

[3]At the time of writing, new learning-based methods are being developed for relightable NeRFs [184]. While plausible relighting may be achieved for opaque hard surfaces, correctly accounting for the many internal scattering interactions in dense volumes seems out of reach of these methods for the time being.

10.2 Conclusion

Combining the algorithmic contributions of Part I and the systems work of Part II, we tackled a challenging application: unbiased inverse rendering of physically based participating media. To complement the differential ratio tracking estimator of Chapter 5, we proposed a simple way to leverage a nonphysical emissive volume model to bootstrap the optimization of scattering volumes, thus avoiding suboptimal local minima.

Combined, our contributions allow the inverse rendering of challenging media and surfaces, recovering high-resolution density and albedo parameters. The result is inherently editable and relightable.

Future work. Our evaluation focuses on synthetic scenes with known camera parameters and illumination. We hope that our contributions can form a building block for further applications, such as the inverse rendering of real-life scenes with imperfect input and using complex appearance models.

Our experiments used a known, simple isotropic phase function. Introducing phase function parameters to the optimization is orthogonal to our method, but can increase the ambiguities and number of local minima. Furthermore, as shown by Zeltner et al. [12] in the context of BSDF optimization, mismatches between the importance sampling distribution derived for the primal and the value of the phase function gradients will lead to significant variance. Here as well, special care must be taken if the pdf can be zero where gradients are nonzero.

In applications where the availability of reference data is limited, such as cloud tomography [134], an interesting direction for future work would be to further utilize the recovered emissive medium used in our initialization scheme. It may act as a proxy model from which imperfect, but infinite reference observations can be generated.

In essence, the proposed initialization scheme post-processes a non-physical model to introduce physical realism. Generalizations of this idea, *e.g.* progressively increasing the maximum path depth over the optimization, should be studied in detail.

Finally, Azinović et al. [128] have found that allocating a high sample count to the primal estimate results in better convergence. While most of our scenes benefited from that same allocation, the challenging cloud reconstruction of Figure 10.1 did not accurately capture the finer detail of the target cloud until at least 512 samples per pixel were allocated to the estimation of gradients (phases II and III). More work is needed to determine the optimal allocation in all scenarios, which will most likely be related to the relative variance characteristics of the primal and adjoint problems.

11 | Inverse Rendering of Real Rooms

Relevant background: Section 2.6 and Chapter 3.

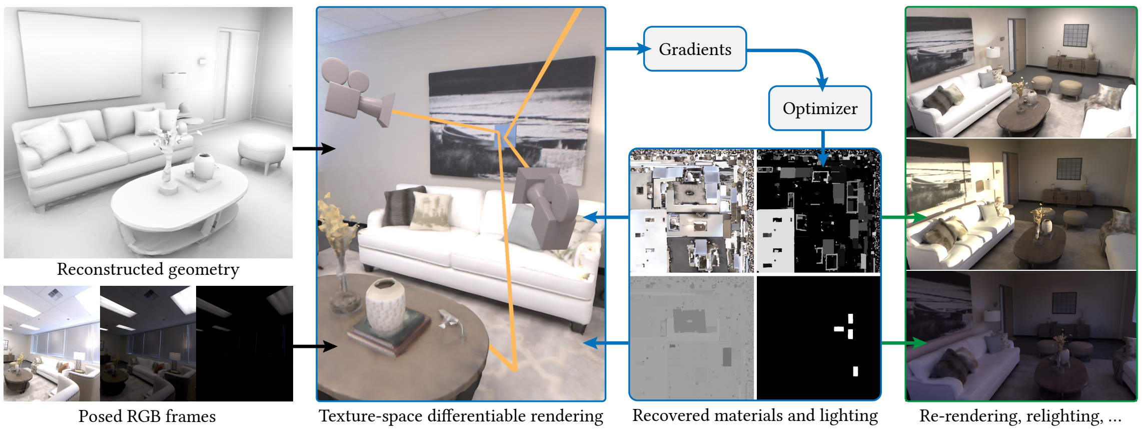

Reconstructed geometry
Posed RGB frames
Texture-space differentiable rendering
Gradients
Optimizer
Recovered materials and lighting
Re-rendering, relighting, ...

Figure 11.1: Given posed RGB frames and scene geometry, our method jointly reconstructs lighting and material parameters of real indoor scenes. Our method relies on differentiable rendering, a new texture space sampling scheme as well as carefully designed inductive priors to achieve high-quality reconstruction at 4K resolution. The optimized material and lighting parameters are readily used in any physically based graphics pipeline, enabling full scene relighting, re-rendering and AR / VR applications.

While Parts I and II were dedicated to building algorithms and systems to efficiently estimate scene parameter gradients, many open-ended steps are necessary to successfully and robustly carry out inverse reconstructions from *real-world data*. This chapter discusses a practical pipeline designed to achieve this goal, including a new sampling technique, inductive priors, and carefully chosen modeling assumptions.

11.1 Introduction

Realistic reconstruction of real 3D environments is a major component of virtual world-building. It enables various simulations, augmentations, as well as augmented and virtual reality (AR/VR) applications, such as virtual object insertion, scene relighting, re-rendering from novel views, and material editing. Computer vision techniques have mostly relied on simple lighting, material and light transport models, that do not account for complex illumination, shadows, or view-dependent reflections (Figure 11.2).

With the recent popularity of inexpensive commodity RGB-D sensors and even mobile LiDARs, incredible advances have been achieved for 3D geometry reconstruction

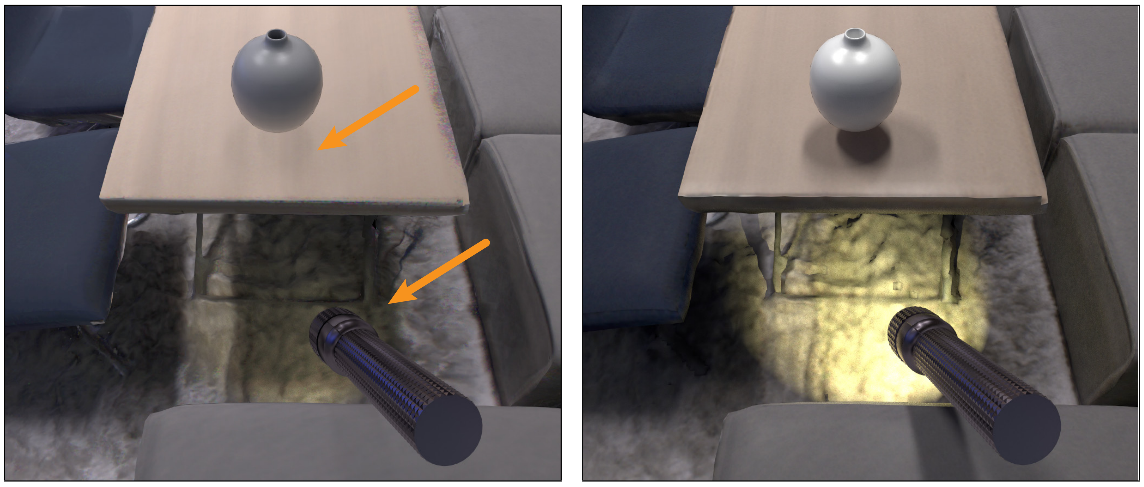


Figure 11.2: Simplified solutions for scene reconstruction (left) typically model the world as emissive surfaces without correctly accounting for light transport, and thus do not support important applications such as virtual object insertion and relighting. Our method (right) recovers emission and material parameters of real-world scenes, which are readily used in photorealistic applications.

[186, 187, 188, 189]. For example, a complex room-scale scene geometry with high dynamic range (HDR) textures and semantic labeling can be fully reconstructed in high quality [190]. However, recovering material or illumination properties requires a deeper understanding of these captured scenes. Limited attention has been devoted to this type of reconstruction, which is a key prerequisite for a seamless photorealistic experience in the aforementioned applications.

Meanwhile, physically based rendering has been successfully used to produce photorealistic imagery for movies, visual effects, and games. In Chapters 3 and 7, we investigated the usage of automatic differentiation to systematically compute derivatives through the rendering process. Leveraging these rendering derivatives, the material and lighting parameters are progressively improved using gradient descent.

Similar to other non-linear inverse optimizations, strong domain-specific inductive bias and a carefully designed optimization routine are key to achieving robust and efficient convergence and high-quality results. For synthetic scenes with perfect geometry and segmentation, the method of Azinović et al. [128] provides high-quality estimates of the scene's materials and lighting. However, only a limited number of proof-of-concept results were shown for real-life captures, with reconstruction degraded quality. As pointed out by the authors [128], imperfections in the input data can have a significant impact on the quality of the recovered materials.

We propose a practical pipeline with several novel priors to robustly handle large real-world captures and address various imperfections in the input data, such as missing reconstructed geometry, camera misalignment and unevenly distributed camera views. In combination with our new texture-space optimization formulation, our method robustly recovers physically based spatially varying materials and lighting in large captured indoor environments, such as the Replica dataset [190]. This chapter's contribution is a joint material and lighting reconstruction method that handles:

- imprecisely reconstructed or even missing geometry;
- large number of unevenly distributed live-captured views, with sensor noise, lens distortion, etc;
- estimation of physically based material parameters (multi-lobe BRDF), including high-resolution (4K) textures.

The output of our pipeline is a physically based scene representation using graphics industry-standard formats, suitable for photorealistic relighting and re-rendering.

11.2 Related work

Beyond the differentiable rendering articles discussed in Section 2.6, we review work specifically relevant to inverse reconstruction here.

Light estimation. Image-based lighting relies on a recovered high dynamic range environment map [191], which represents the illumination incident from every direction at a single point in the scene, to realistically relight virtual objects at that location. The recovered environment map can be further approximated with certain basis functions, such as spherical harmonics [96, 192] or spherical Gaussians [193], for efficient rendering. Convolutional Neural Network-based approaches can also automatically estimate an environment map from a single indoor image [194]. Other methods [193, 195] attempt to reconstruct local spatially-varying lighting from a single image. Gardner et al. [196] utilize a deep encoder network to recover parametric lighting in the scene. In our method, lighting is represented locally as emission from surfaces that are physically present in the scene, e.g. neon lights on the ceiling. The position and radiance of those emitters are determined automatically through our optimization process.

Material recovery. Chen et al. [197] and Kang et al. [198] cast reflectance capture into a form that admits a solution via deep networks. Encoder-decoder architectures are used for appearance capture and rendering of human faces [199], image-based relighting from sparse samples [200], and appearance maps [201]. Deschaintre et al. [122] use a differentiable re-rendering loss and procedurally generated materials to train a deep network recovering SVBRDF parameters. Gao et al. [202] optimize directly in the latent space learned by an auto-encoder, which acts as a regularizer. For more details, we refer to the survey of Dong [203] on deep appearance modeling. To the best of our knowledge, none of the existing methods handle non-Lambertian materials, spatially-varying *local* illumination and global light transport all at once.

Material and shape recovery. Schmitt et al. [204] rely on a hand-held RGB-D scanner with active illumination to the reconstruct the geometry and SVBRDF of a single object. They use differentiable material clustering to improve the estimation of specular components. Several recent works [205, 206, 207] use deep cascaded architectures trained on synthetic datasets to recover the shape and microfacet SVBRDF of a single object from one or two handheld pictures. Li et al. [207] additionally account for global illumination with dedicated neural blocks. In contrast, our method scales to large indoor scenes with significant interreflection.

Joint estimation of material and lighting. Barron and Malik [208] recover geometry, reflectance, and illumination from a single image of an arbitrary object by enforcing hand-crafted priors on each component. Li et al. [193] similarly recover depth, SVBRDF and local illumination from a single viewpoint using a deep network trained on realistic synthetic interior scenes. Karsch et al. [209, 210] render synthetic object into real photos. Zhang et al. [106] achieve plausible results in recovering the reflectance of walls, floor, and ceiling of indoor scenes along with lighting using inverse rendering.

Azinović et al. [128] took a step toward the general use of differentiable rendering for reconstruction, though it remains far from achieving this goal for real captures with imperfect input data. Our method builds on this approach while supporting complex spatially-varying materials that are reconstructed from real captured data.

Intrinsic decomposition. Image-space methods [122, 208, 211, 212, 213] employ sophisticated data-driven approaches, by learning the distributions of material and illumination. However, these methods do not have a notion of 3D geometry, and cannot handle occlusion, interreflection, and physically based factors such as the squared distance falloff of light intensity. They also require a significant amount of training data, and are prone to errors outside of the training data set.

Deferred neural rendering. Deferred neural rendering [214] achieves novel view synthesis, scene editing, animation synthesis, or free viewpoint relighting [215] by optimizing a neural texture jointly with a neural renderer. The high-dimensional neural texture, mapped to a simple 3D proxy surface, is sampled as in the standard graphics pipeline to produce features that are decoded into an image by the neural renderer. In contrast, our method produces physically based textures that are readily used in traditional renderers.

Stratified sampling. Stratified sampling [23], discussed in Section 2.1.7, is a well-known technique used in Monte Carlo rendering to reduce variance over purely uniform sampling. In the context of non-line-of-sight geometry reconstruction, Tsai et al. [36] sample over the surface of the recovered shape, in order to improve rendering efficiency (fewer missed rays) as well as to produce more coherent ray bundles. In the same spirit, our texture space sampling technique improves convergence by sampling uniformly in the space of optimization variables rather than generating rays in camera space.

11.3 Method

In the same fashion as the caustic design and inverse volume rendering applications of Chapters 9 and 10, our method uses an analysis-by-synthesis approach. At each step, images of the current state of the scene are produced using differentiable rendering algorithms and compared to the observed reference. The difference is then minimized using gradient-based optimization. Figure 11.3 shows the high-level pipeline discussed in the remainder of this section.

Unlike similar methods based on rasterization, our rendering step builds on a differentiable path tracer and physically based appearance and illumination models. The resulting images account for global illumination, which is a prerequisite for high-fidelity parameter reconstruction, as illustrated in Figure 11.4.

11.3.1 Input data

The input to our method is a reconstruction of the scene geometry and a set of RGB photographs with camera intrinsics and extrinsics ("posed frames"). We further use an approximate segmentation of the scene's surfaces, which can either be computed automatically or provided as input. Our method reconstructs suitable material and lighting

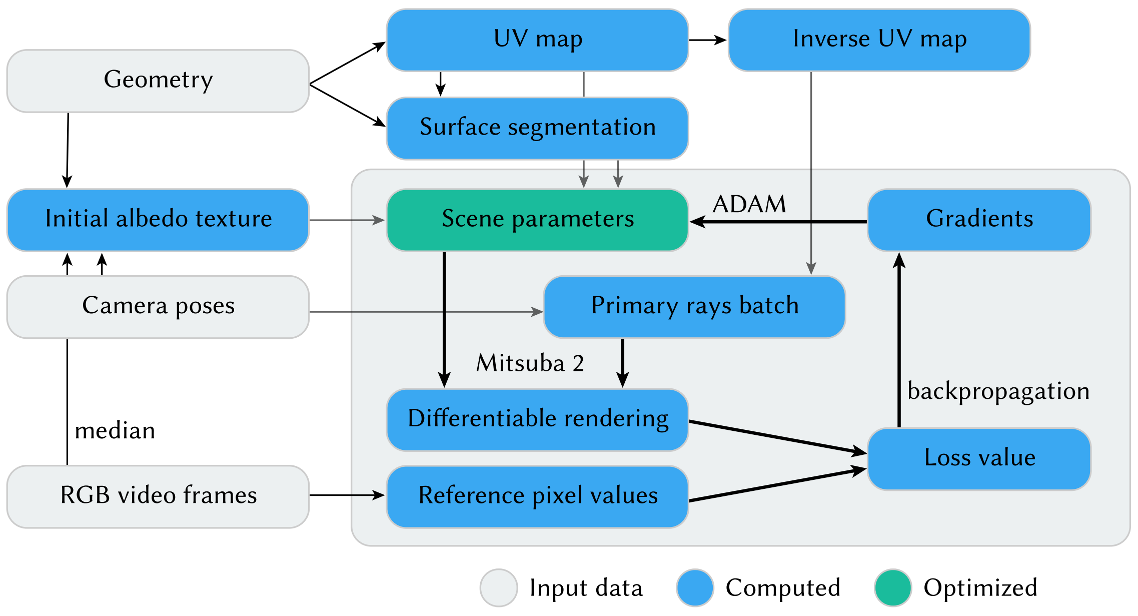


Figure 11.3: Our reconstruction technique uses differentiable rendering to recover complex spatially varying materials and light sources from posed handheld RGB frames. A novel texture-space sampling scheme robustly handles uneven coverage and imperfections in real-world data.

parameters, but does not modify the input geometry. Joint material and geometry reconstruction is in principle possible [57, 112, 113], but adds implementation complexity and runtime costs, and is therefore out of the scope of this work.

The Replica dataset. We use the *Replica* dataset [190] in our examples, which consists of multiple indoor scenes acquired using a handheld device. The capture was performed by an operator walking through the scene, holding a specialized rig. The resulting data was processed using standard methods producing camera extrinsics and intrinsics, a triangular mesh of the scene, and an approximate instance segmentation. As with any real-life capture, video frames must be corrected for camera calibration: color shift, vignette, and distortions are removed in a preprocessing step.

The dataset also includes a fused high-dynamic range (HDR) texture for each scene. However, unlike our method's results, that texture has all shadows, view-dependent effects and global illumination "baked-in" and is therefore not suitable for relighting, scene editing, realistic re-rendering with correct view-dependent effects, etc.

Dyanamic range. Our method is not specific to the Replica dataset. That being said, physically-based material and lighting reconstruction does require a sufficient dynamic range of observations, *e.g.* to provide some direct observations of light sources and high-

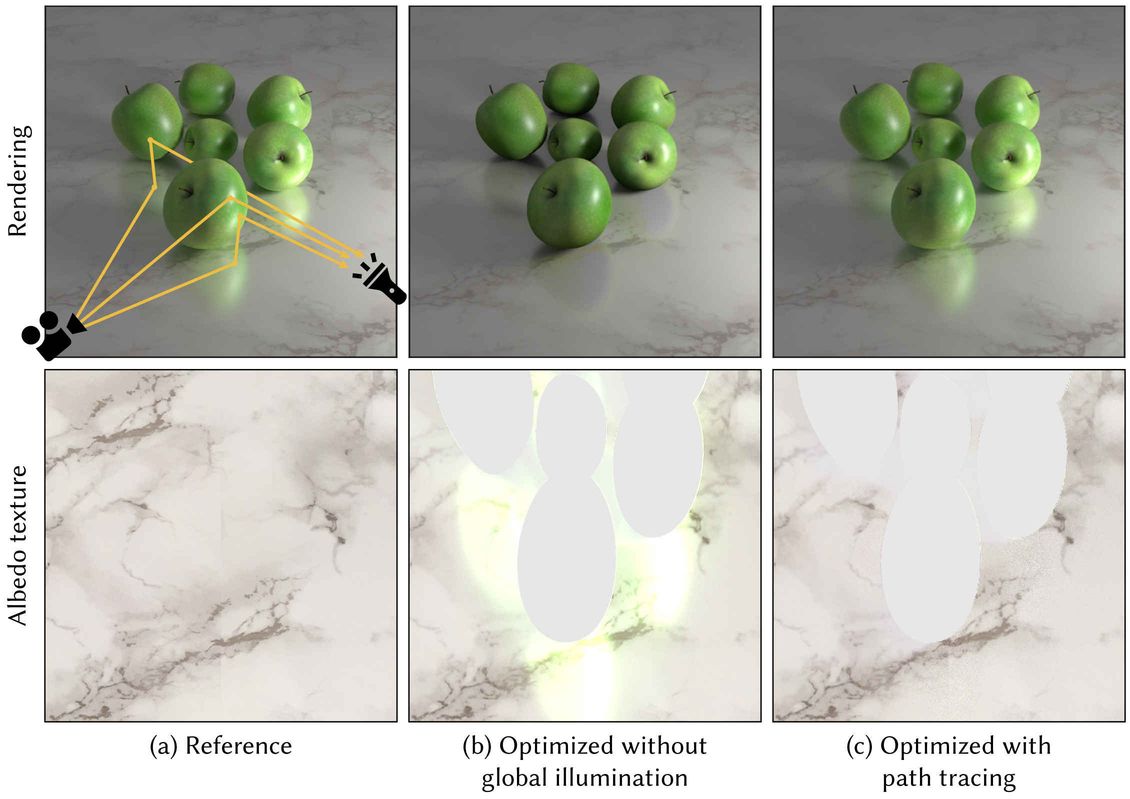

Rendering
Albedo texture
(a) Reference
(b) Optimized without global illumination
(c) Optimized with path tracing

Figure 11.4: Real-world light transport features global illumination (GI) between objects. Reconstructing the marble table texture in this synthetic scene **(a)** poses severe challenges for classic techniques. The reflection of the apples cannot be explained without GI **(b)**, and this discrepancy between reality and reconstruction can only be explained via incorrect color bleeding into the texture. A differentiable path tracer **(c)**, such as the one discussed in Chapter 4, can disentangle the effects caused by individual objects. In both cases, unobserved texture regions remained at their initialization value (gray).

lights without overexposure. In the Replica captures, one of the sensors is an RGB camera that records a video stream in Standard Dynamic Range (SDR). Extended dynamic range is achieved by cycling exposure time every frame (*multiplexed HDR*), as shown in Figure 11.5. Therefore, there is no single HDR image available for a specific viewpoint, but under- and over-exposed areas change at each frame.

Handling imperfect input data. Regardless of the source, real-world data invariably contains noise and imperfections such as imprecise camera poses, surface normals, missing fine geometry, and inexact segmentation, producing systematic discrepancies between renderings and the observations. It is imperative that the method handles such flaws gracefully.

In our pre-processing step, we discard frames with severe under- and over-exposure

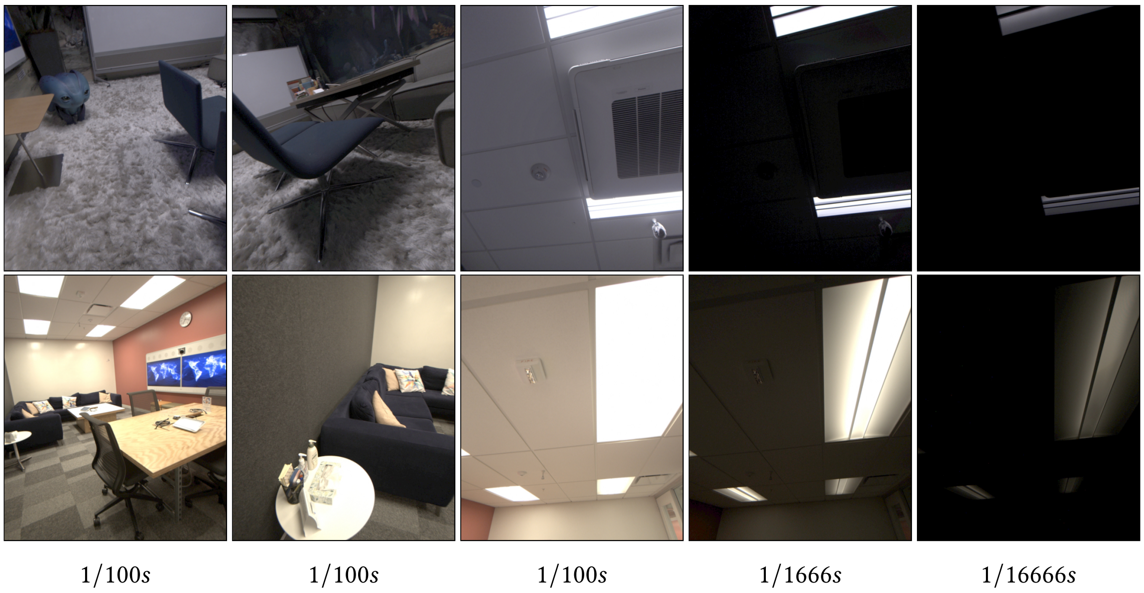


Figure 11.5: Scenes from the Replica dataset [190] were captured with a handheld camera rig. The RGB video stream uses multiplexed HDR: SDR frames alternate between three exposure times. For each camera pose, we therefore have access to a single SDR frame only.

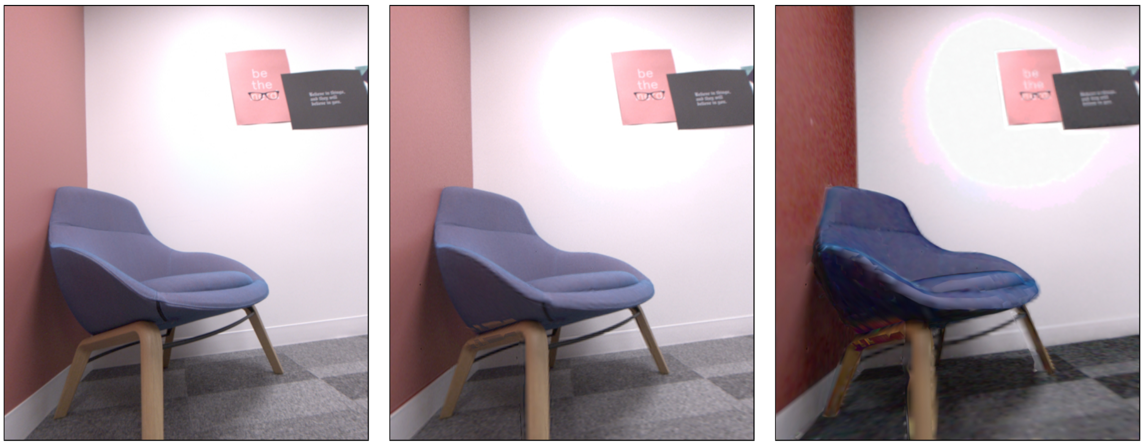
and motion blur, which are easily detected from the difference in camera pose between adjacent frames. We also linearize and white-balance the images, and remove lens distortion. Scene geometry is represented with a standard triangle mesh with UV texture parametrization. The original Replica data lacks UV coordinates, so we generate them automatically using Blender's standard "Smart UV Project" operator [1].

11.3.2 Inductive bias & modeling assumptions

The inverse problem targeted by our method is highly ambiguous: each surface location within the scene can in principle affect the color of any other position via indirect reflections. Because the light emitted by a light source can interact with multiple materials before arriving at the camera, any given observation can be explained in multiple ways. For example, objects seen via specular reflection can be misattributed as emission or diffuse reflectance (Figure 11.4).

Therefore, a naive application of image-based differentiable rendering systematically overfits with poor local minima. We introduce several inductive biases and heuristics that promote plausible and consistent results to address these issues.

Emitters. We model light sources as area lights (emissive surfaces) with a cosine-weighted directional profile. However, emission is difficult to disentangle from reflec-



(a) Reference (b) Optimized (re-rendering) (c) Optimized (emission only)

Figure 11.6: Unconstrained joint optimization of material and emission reproduces the reference **(a)** with high fidelity, as shown in **(b)**. However, the solution found by the optimizer is absurd, since it turns the entire scene into an emissive surface **(c)**. Since no light sources are visible in that frame, a correct reconstruction would have **(c)** be entirely black.

tion. For example, a highlight observed on a surface can be misinterpreted as a light source. In our experiments, allowing unrestricted optimization of emission values always converges to implausible solutions, with nonzero emission values on all scene surfaces. The problem is illustrated in Figure 11.6.

Therefore, we initially restrict spatial variation of emission to a single intensity value per object, based on the instance segmentation. Once light sources have been identified by the optimization, we enable spatially-varying emission over those regions. Here, there's an implicit assumption that light sources are significantly brighter than non-emissive surfaces.

Material model. Recall that the bidirectional scattering distribution function (BSDF) models the surface material and defines how much radiance is reflected or refracted from an incident direction to an outgoing direction (Equation (2.20)). A large variety of general and specialized BSDF models have been proposed throughout decades of research [216].

Since we do not assume access to a classification of materials over the scene, we choose a single material model that can cover the majority of appearance, while keeping the number of parameters to a minimum. Importantly, we do not support transparent or refractive materials. Spatial variations are handled by texturing the model's parameters over the scene's surfaces.

The Disney BRDF [217] is a widely adopted material model used in movies and games

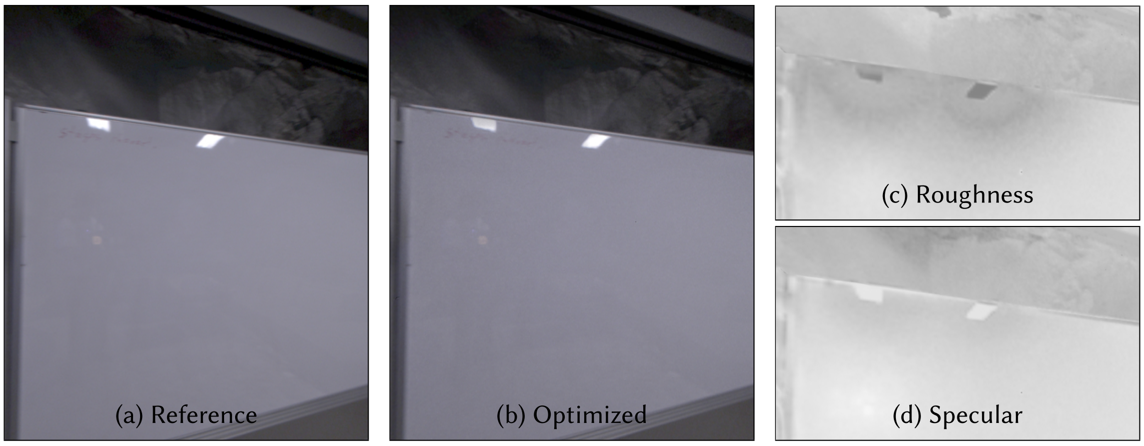


Figure 11.7: Unconstrained optimization of material parameters leads to implausible results, such as materials being highly glossy *only* where highlights were observed.

that captures a versatile set of appearance with intuitive controls. It exposes ten high-level parameters such as base RGB color, metalicness, roughness, and clearcoat. In unstructured optimization, however, different parameter configurations often lead to similar appearance (underdetermined problem). To reduce this ambiguity, we restrict optimization to only handle opaque surfaces with diffuse albedo, roughness, and specular parameters.

A second important ambiguity is due to the fact that we can judge a surface's roughness at a point only when we observe an actual highlight at that point. Therefore, the optimizer is able to infer surface specularity and roughness *only where* the specular highlights are observed (Figure 11.7). However, we cannot expect our captured data to observe highlights at every point where they could occur. Therefore, we assume the roughness and specularity to remain constant within a class of the surface segmentation and optimize a single roughness & specularity value per object. This is a common case in the real world and also reduces the number of parameters to optimize, speeding up convergence.

Here, the underlying assumption is that the segmentation is sufficiently granular to separate surfaces with different materials, even on the same object.

Initialization. Proper initialization is important to guide optimization toward a good minimum. We start with neutral constant values for the emission (0), roughness (0.5) and specular (0.5) parameters. To initialize the spatially-varying diffuse color, we build a median texture by iterating over all available reference pictures and projecting them in texture space. We employ an online median estimator [218] to avoid memory concerns when given tens of thousands of views. We selected a median rather than a mean filter to suppress view-dependent effects such as specular reflection.

11.3.3 Texture-space sampling for variance reduction

In Chapters 9 and 10, we used a straightforward image-by-image optimization loop: the current state of the scene is rendered from one or multiple sensors, and we minimize the distance to the corresponding references. All reference pixels are given equal importance. However, the target unknowns (material and illumination parameters) lie in the space of scene surfaces (texture space). Therefore, in the current application, optimizing over all pixels within a frame(s) leads to uneven convergence. Indeed, the unknowns (e.g. albedo texture values) are observed unevenly within a single view, as well as over the video sequence. This is visualized in Figure 11.8. Additionally, view-dependent effects such as glossy highlights require multiple observation angles to disambiguate the roles of diffuse and specular components.

Texture-space sampling. Instead, we propose to form the training batches by sampling the unknowns uniformly *directly* in texture space. This is more efficient than sampling a random subset of views, as it allows to reduce the noise in gradients by proceeding with batches of observations that are directly relevant to the selected unknowns.

Texture-space sampling is realized as follows. At the start of each iteration, we select a subset of the unknown variables by sampling uniformly at random over texture space. The number of sampled points effectively determined the batch size, and can be adjusted based on the available GPU or system memory. Using a precomputed inverse UV mapping, we look up the corresponding 3D positions on the scene surfaces. Next, we connect the sampled mesh positions to a random set of reference view positions. Connections that are occluded by geometry, or simply fall outside of the cameras' frusta, are discarded. For this batch of visible 3D positions, we fetch the corresponding pixel values from the reference RGB frames. Finally, we estimate the current radiance values for the batch with differentiable path tracing. After computing the per-pixel loss (averaged over all rays), gradients are obtained by backpropagating through the rendering algorithm. Finally, scene parameters are updated with an optimizer step.

Jacobian factors. Transformations applied to the samples–mapping from UV space to scene surfaces and finally to camera rays–imply a change of probability density. In standard Monte Carlo rendering, that change should be accounted for when computing

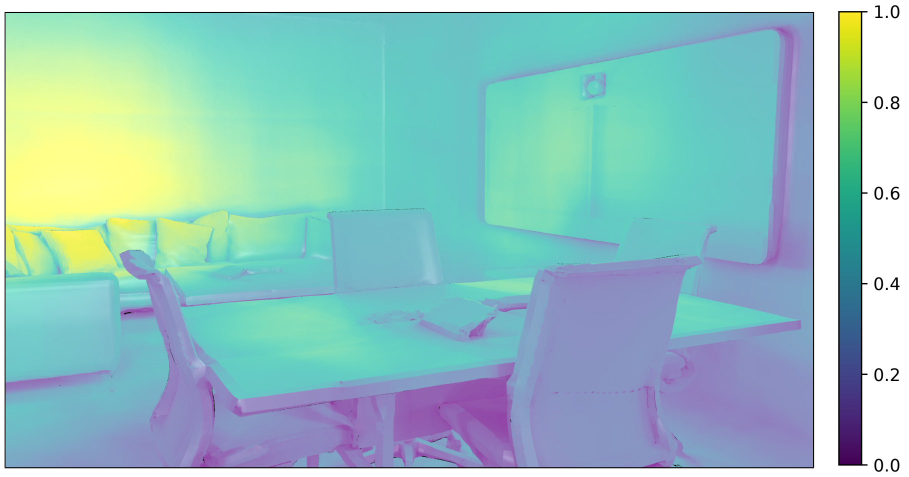


Figure 11.8: Unstructured capture of real-life scenes from handheld video results in an uneven density of observations, which in turn leads to uneven convergence with naive inverse rendering. We visualize the relative number of observations for each texel of the OFFICE-2 scene [190].

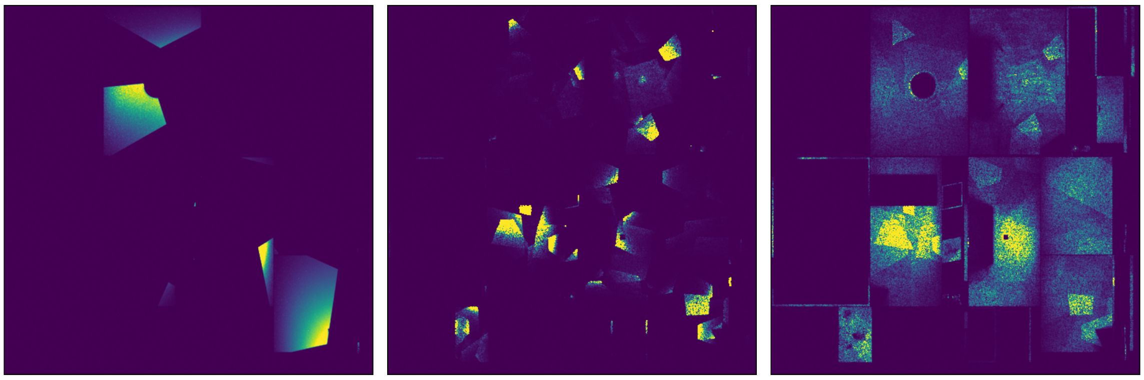
the sampling weight by multiplying it with the Jacobian determinant of each transformation [19, 24]. These factors include a term accounting for the UV mapping's distortion, as well as a geometry term $\frac{\cos(\theta)}{d^2}$, where θ is the incident angle to the sampled surface and d is the distance to the camera. Intuitively, the geometry term corrects for the fact that sampling that same surface point from the camera's directional distribution becomes less likely as the distance increases, or the observation angle more grazing.

However, the explicit goal of our sampling technique is to assign equal weight to all optimization variables. Note that in the context of an optimization, we are free to define the objective function as needed to improve convergence and reconstruction quality. To this end, we omit the Jacobian terms above, implicitly introducing a factor α_j canceling them out in our per-ray objective function.

This is in contrast with the method of Tsai et al. [36], where reconstructed surface points are sampled directly and all Jacobian terms are included. In their non-line-of-sight reconstruction application, observation distances are roughly constant, while ours vary greatly from viewpoint to viewpoint. Omitting Jacobian factors also helps us avoid exploding gradients when d approaches zero. Finally, we obtain gradients of comparable magnitude spread evenly over texture space, as illustrated in Figure 11.9.

11.3.4 Optimization details

We found that several low-level details of the pipeline had a significant impact on the reconstruction quality. We summarize them here and refer to the original article [16] for



(a) Image-space (b) Texture-space (c) Texture-space (re-weighted)

Figure 11.9: We illustrate the gradients produced by different sampling techniques. We visualize albedo gradient magnitude at a given iteration, directly in texture space. The standard image-based optimization loop **(a)** leads to nonzero gradients only within the frusta of cameras selected for this iteration. Our texture-space sampling scheme **(b)** selects texels uniformly at random and connects them to camera positions, resulting in even coverage within an iteration. Finally, we implicitly reweight the objective function by omitting Jacobian factors **(c)** in order to obtain gradients of comparable magnitude regardless of observation distance or angle. Note that some locations are not mapped to any scene surface and thus have zero gradients.

the complete list.

Coarse-to-fine optimization. Due to the ambiguities described in Section 11.3.2, starting the optimization with all unknowns of a large scene at their highest resolution leads to low-quality local optima. "Coarse-to-fine" schemes have been shown to help greatly in previous work [99] and in our own experience (Chapters 9 and 10). The albedo is first optimized as a 1024×1024 RGB texture, and then refined in two stages to reach the final 4096×4096 resolution. After the first stage, emitters contributing a small fraction of the total scene radiance are rounded to zero. In the same spirit, we gradually introduce optimization variables: first emission, then roughness & specular coefficients, and finally spatially-varying albedo.

Loss function. We use a pixel-wise mean squared loss. The loss operates in linear color space, i.e., without gamma compression. Under-exposed (resp. over-exposed) values are handled with a one-sided difference that only penalizes values above (resp. below) the clipping threshold v_{min} (resp. v_{max}).

Optimizer. We use the Adam optimizer [150] with learning rates 1 for emission, 0.005 for roughness & specular, and 0.1 for diffuse albedo. We found two additional modifications to be helpful.

First, recall that in addition to the noise from stochastic gradient descent (only a small subset of all reference values are optimized at each step), each path-traced sample is itself a noisy Monte Carlo estimate. The path-traced sample values often include high-valued outliers caused by improbable light paths. We minimize their impact by clamping gradients to $\pm 10^{-8}$ (before Adam rescaling) to prevent these outliers from contaminating the optimized textures. While this introduces bias, we found that it greatly improved convergence overall.

Another important observation is that at each iteration, only a subset of the scene is observed and can receive meaningful gradients. Moreover, the moment estimates maintained by Adam inevitably include Monte Carlo stochastic estimation noise from previous iterations' gradients. As a result, at each step of the optimizer, noisy momentum is applied repeatedly to all variables, *even those not observed in the current batch.* As long as no new observations are made for a given variable, the noise pattern in its momentum remains fixed, impeding convergence. To alleviate this issue, we restrict the application of momentum within the optimizer, as well as updates to the moments, to variables that receive nonzero gradients at that iteration[1].

Discarding indirect gradients. In the indoor scenes of the Replica dataset, we found gradients for indirectly-observed parameters to be extremely noisy due to the low sampling probability of long light paths. Understandably, an observation of a diffuse wall tells us relatively little about the opposite wall, even though some indirect light has likely come from there. When the available data allows, it is preferable to rely on direct observations, in order to minimize variance. Therefore, we exclude indirect light bounces from the gradient computation. This results in memory and computational savings[2] as well as faster convergence due to the reduced noise in gradients. The idea to voluntarily exclude a noisy gradient term to improve overall convergence behavior is reminiscent of the biased radiative backpropagation variant of Section 4.3.8.

Note that we still compute the correct path-traced solution, so global illumination is fully accounted for: the net effect is simply to restrict gradient-based updates to regions

[1]A similar technique is employed when optimizing over sparse parameter tensors, implemented for example in PyTorch's `SparseAdam` optimizer [159].

[2]If using radiative backpropagation instead of an AD-based implementation, these tradeoffs may balance differently, as the memory footprint would no longer be the main limiting factor.

that are directly observed in a given iteration, while disentangling indirect effects.

Averaging of iterates. Optimization eventually wanders around the true solution due to Monte Carlo noise in the rendered images. We apply Polyak-Ruppert averaging [219, 220] by maintaining a running average of the parameter values over the last 10% of the optimization.

11.4 Results

We now evaluate our method on challenging real-world scenes and showcase applications such as full scene relighting and seamless scene editing. Evaluation against the method of Li et al. [193] and an ablation study are available in Appendix B. Additional results, including animated sequences, validation on synthetic data, and a study of sensitivity to the input data quality are available in the original article's supplementary document and video [16], which can be accessed at:

`https://rgl.epfl.ch/publications/NimierDavid2021Material`

11.4.1 Reconstruction of real captured scenes

We apply our reconstruction pipeline to the *Replica* dataset [190], which includes reconstructed geometry, an approximate instance segmentation, and posed reference images captured with a handheld rig. This input has imperfections, including noisy camera registration and missing detailed geometry, which make robust reconstruction challenging.

In order to compare against the captured ground truth, we re-render the scene from known viewpoints after the optimization has been completed. Our re-rendered images match the captured frames closely, including fine textured details and view-dependent effects, as shown in Figure 11.10. The reconstructed scenes do not overfit the training views, as shown in the out-of-distribution re-renderings, animated results and comparisons given in the supplemental video[3].

11.4.2 Implementation

Our method was implemented in Mitsuba 2 (Chapter 7), specifically using the automatic differentiation-based backend. The availability of AD made it easy to experiment

[3]The supplemental video is accessible at:
`https://rgl.epfl.ch/publications/NimierDavid2021Material`.

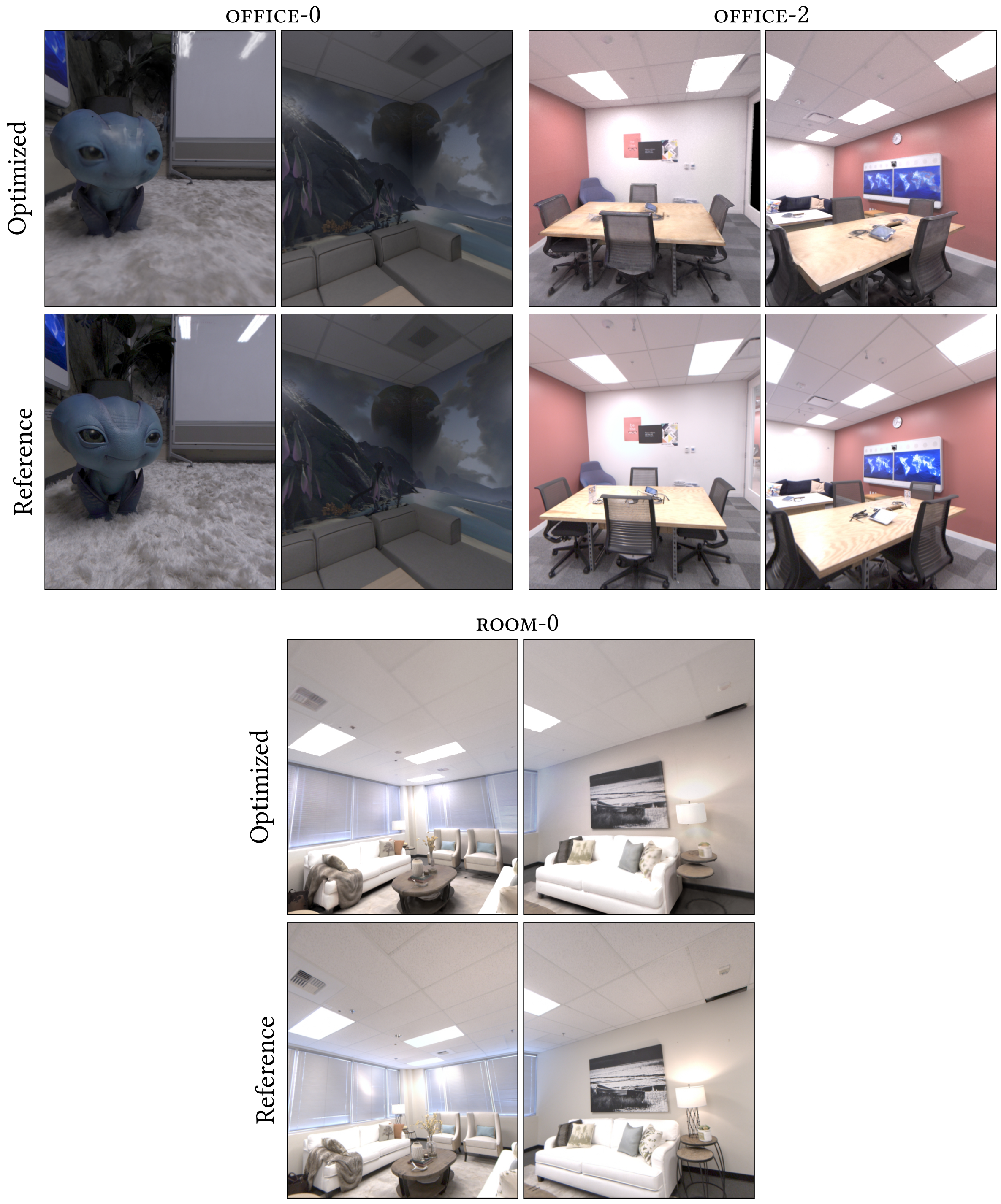


Figure 11.10: Re-rendering scenes from the Replica dataset [190] using the materials and emission parameters recovered by our method matches the reference images closely, including view-dependent effects and high-frequency detail.

with different combinations of BRDF lobes, representations for scene components, loss functions, etc. Nevertheless, our method could be implemented within any framework as long as the relevant derivatives are computed. Each scene reconstruction ran for 12 hours on average on a single NVIDIA Titan RTX GPU within our experimental codebase.

Note that an implementation in Mitsuba 3, using radiative- or path replay backpropagation (Chapter 4), would exhibit much faster runtime[4]. Several parameters such as the batch size and total iteration count could be increased, which we expect would in turn improve reconstruction quality.

11.4.3 Applications

Scenes reconstructed with our method generalize well to out-of-distribution views and can be rendered from any previously unobserved viewpoint (Figures 11.12 and 11.13). We additionally visualize our method's outputs: a set of textures representing the scene's emission and physically based material parameters (diffuse albedo, roughness, and specular). The reconstructed albedo textures are shown in texture space in Figure 11.11 They are well disentangled and noise-free despite significant Monte Carlo noise present during optimization and dataset imperfections.

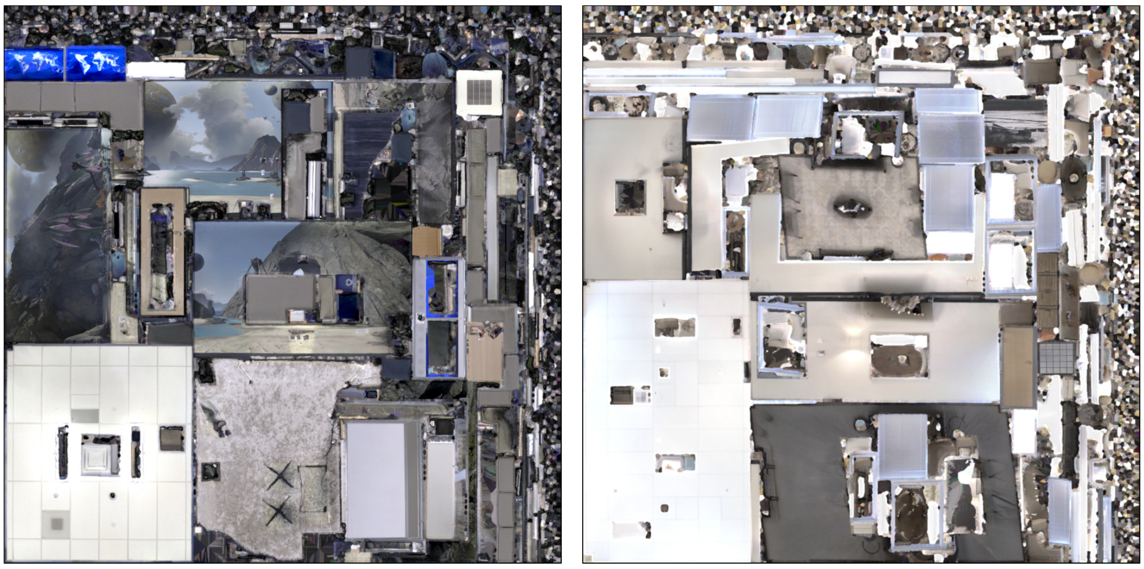


Figure 11.11: Our method uses texture space sampling to uniformly sample the scene's parameters during optimization. We visualize the optimized albedo texture of scenes OFFICE-0 and ROOM-0 in texture space (i.e. unwrapped UV space). The 4K textures were downsampled before embedding into the document.

In this format, the scene is ready for use in standard rendering pipelines for photo-

[4]These algorithms and systems were not ready for use at the time this method was being developed.

realistic applications such as scene editing, novel view synthesis, and relighting. Figure 11.14 (left) demonstrates embedding four virtual objects in the scene, which is a common task for mixed reality applications. Without any additional processing or manual work, the inserted objects blend in and interact correctly with their surrounding (e.g., see reflections, shadows, matched lighting).

Figure 11.14 (right) show a complete re-lighting of the scene: existing illumination was removed and a brightly colored light source was added near the floor. Likewise in Figure 11.15, the existing emitters were replaced by an environment map emulating the daylight cycle. In both cases, the scene reacts correctly to the new illumination, and there are no visible residuals of the original illumination the scene was captured in, such as baked shadows or highlights. Note that this would not be possible without correctly disentangling materials and lighting, as shown in Figure 11.2.

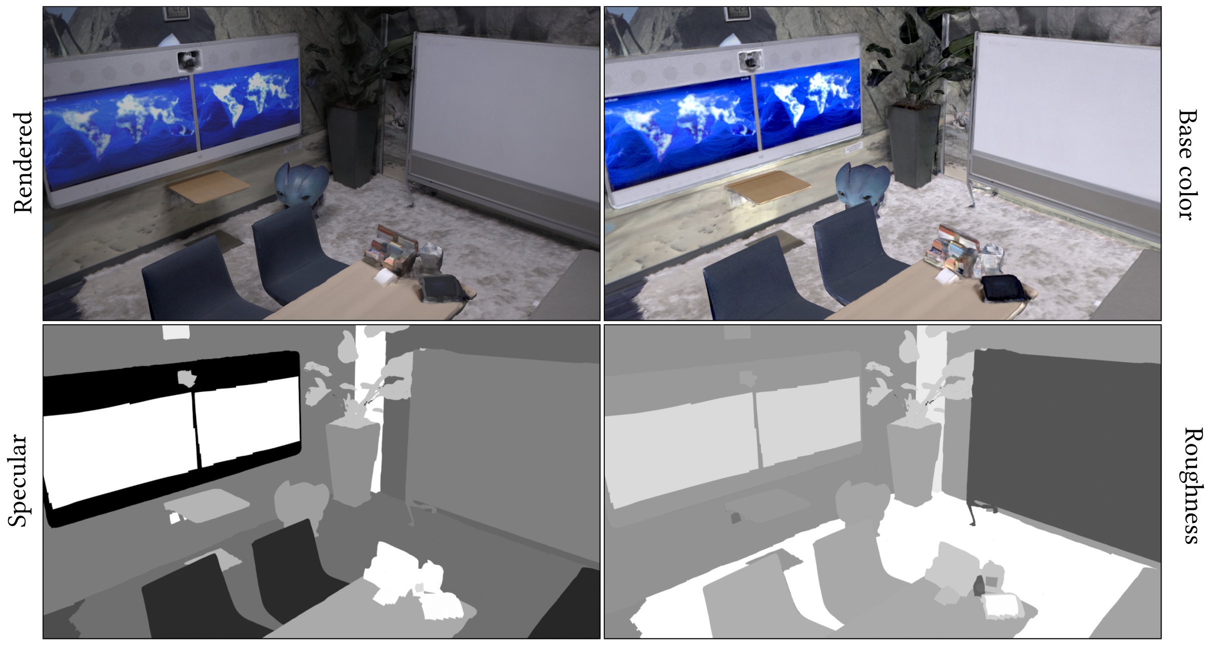
We believe these and many other possible applications enable more seamless integration of real and virtual worlds in scenarios like virtual and augmented reality, robotics simulation, and dataset augmentation.

11.5 Conclusion

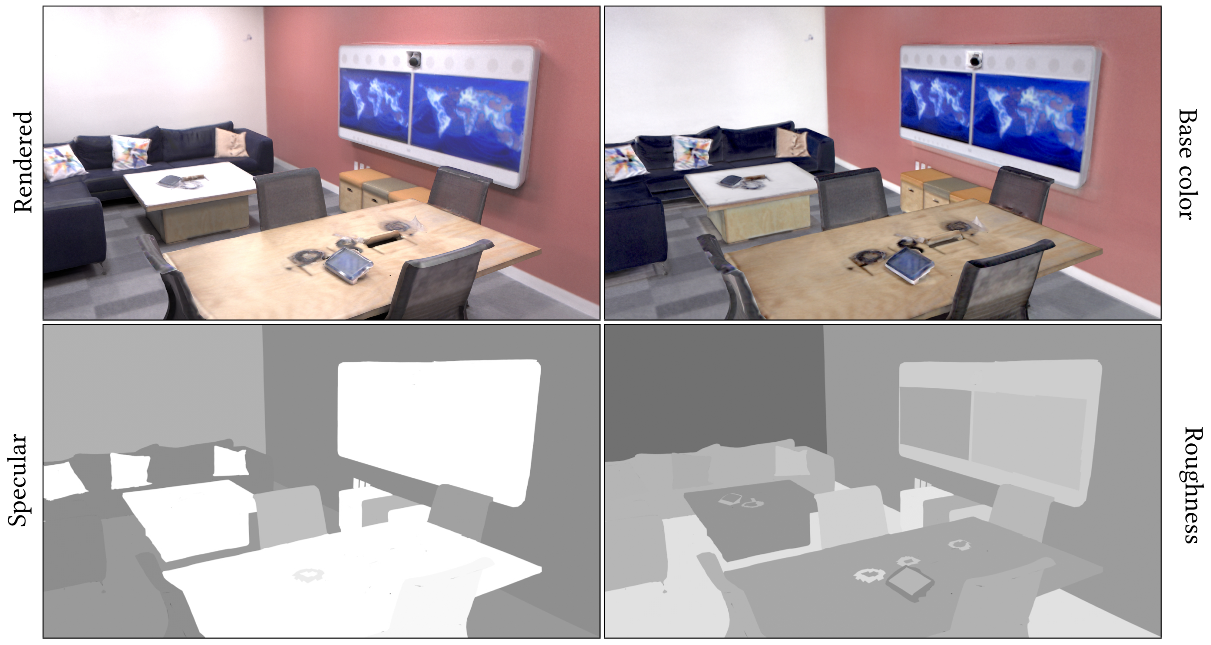
We presented a method for material and lighting reconstruction in large captured environments based on the differentiable rendering framework of Chapter 7. In order to gracefully handle the unavoidable capture & inputs imperfections, uneven coverage of the reference images, as well as correctly disentangling spatially-varying material and illumination parameters, we introduced a texture-space optimization scheme, carefully chosen inductive biases and heuristics, which guide the reconstruction toward high-quality minima.

Limitations. The main limitation of our method is its reliance on the input reconstructed geometry. In our current scene parametrization, rendered images may only explain observations where geometry is present. For example, if a highly emissive or specular object is entirely missing from the geometry, its contribution will most likely be outprojected and attributed to the background objects by the optimizer. This behavior can be observed in the reconstruction of ROOM-0 (Figure 11.13, top). Automatically detecting and adding missing emitters would be a valuable improvement in future work.

Our method does not currently handle reconstruction of transparent objects, even if their shape is correct in the provided geometry (which is a challenge in itself). An

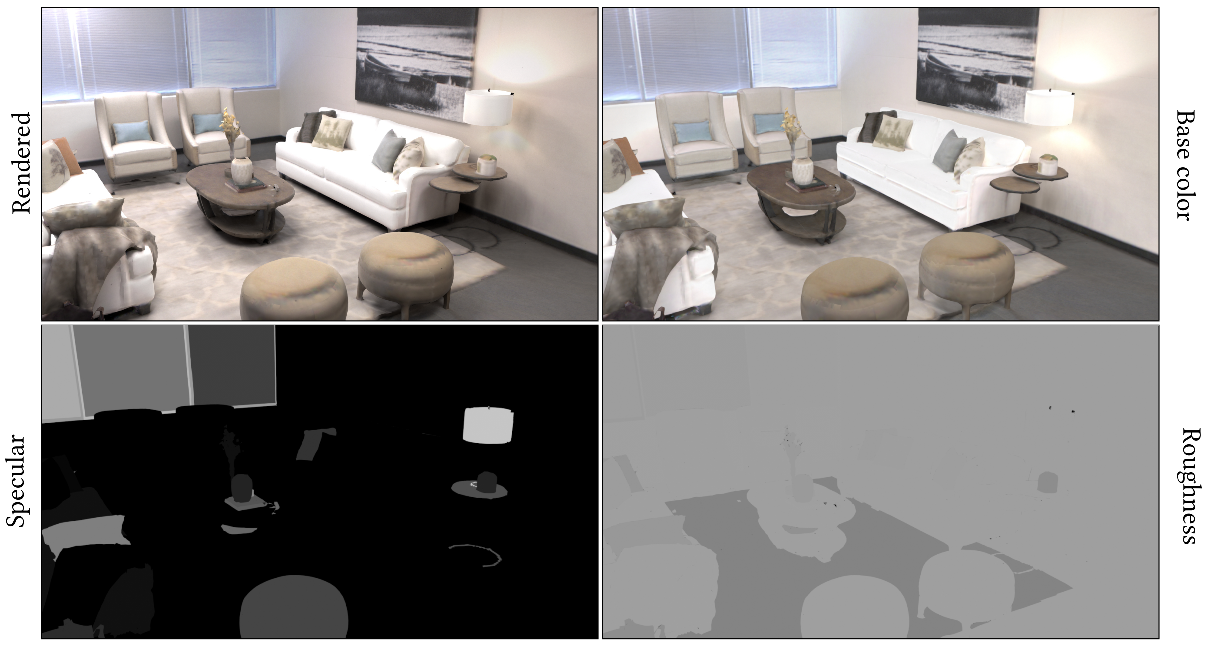


(a) OFFICE-0 scene

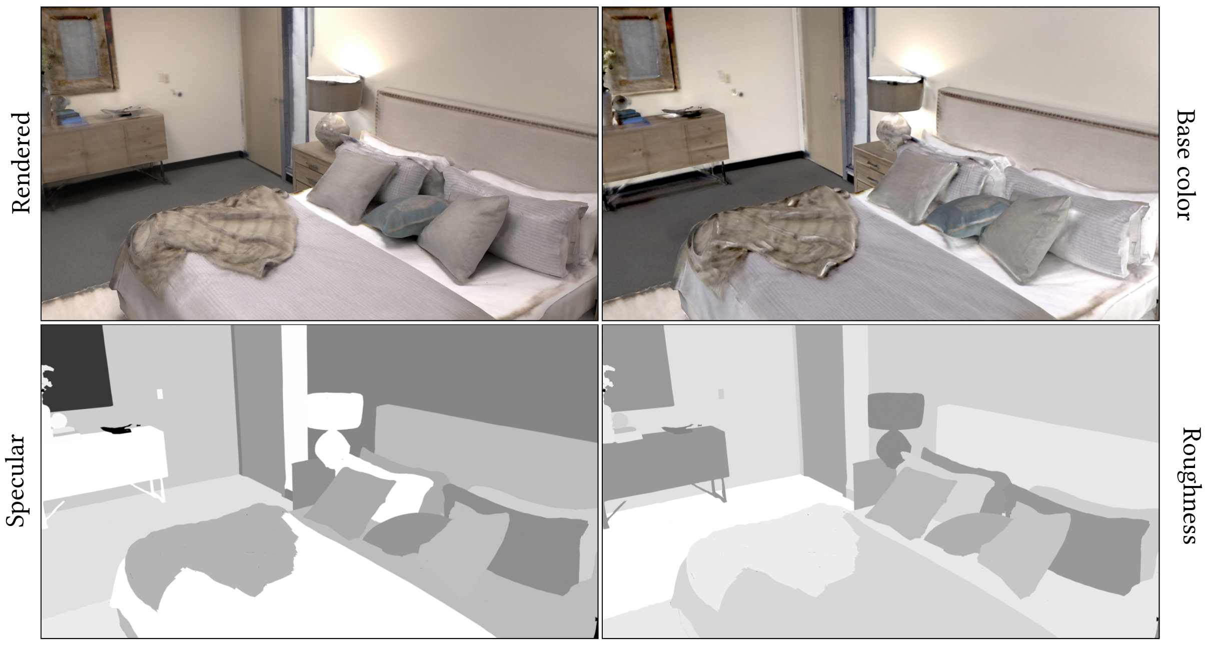


(b) OFFICE-2 scene

Figure 11.12: Scenes obtained with our method can be re-rendered from any viewpoint. The base color texture includes fine detail (4096 × 4096 resolution) and all recovered parameters are physically based and correctly disentangled. *Continued in Figure 11.13.*



(a) ROOM-0 scene



(b) ROOM-1 scene

Figure 11.13: Novel viewpoint rendering, continued from Figure 11.12.

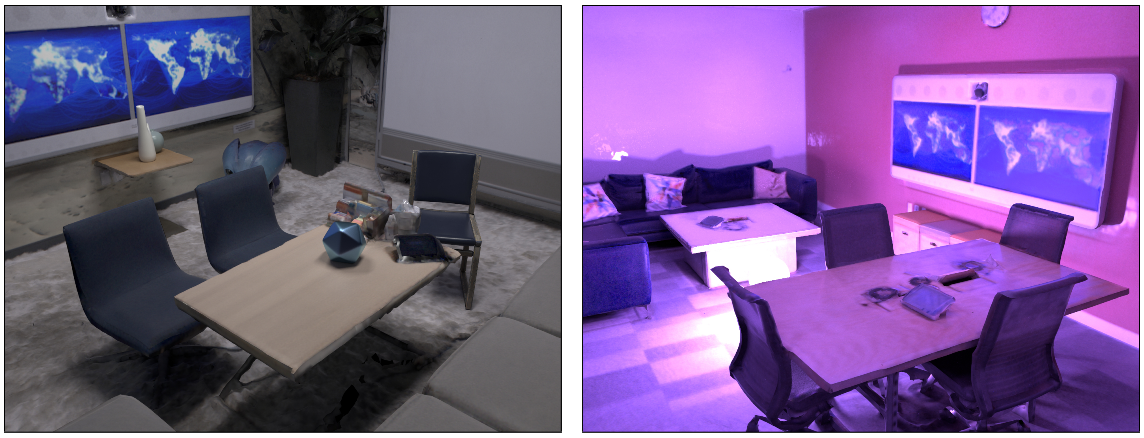


Figure 11.14: Using our recovered emission and materials, adding virtual objects to the OFFICE-0 scene (left) automatically results in correct shadows, reflections and indirect illumination. Scenes can additionally be relit with arbitrary light sources and rendered from any viewpoint (right).

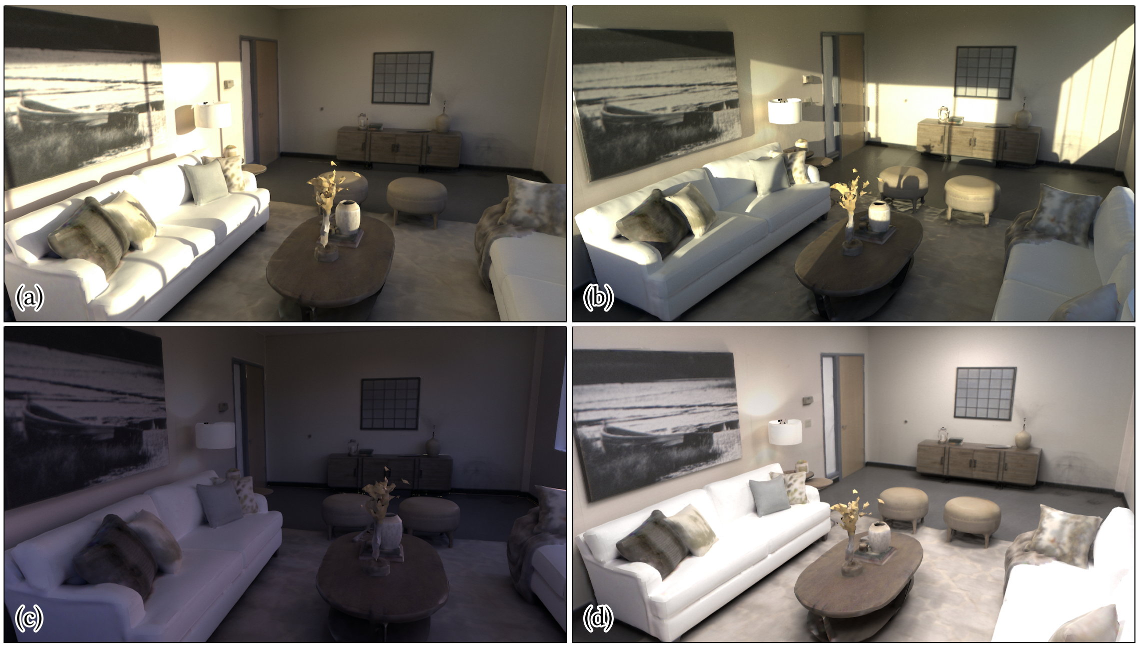
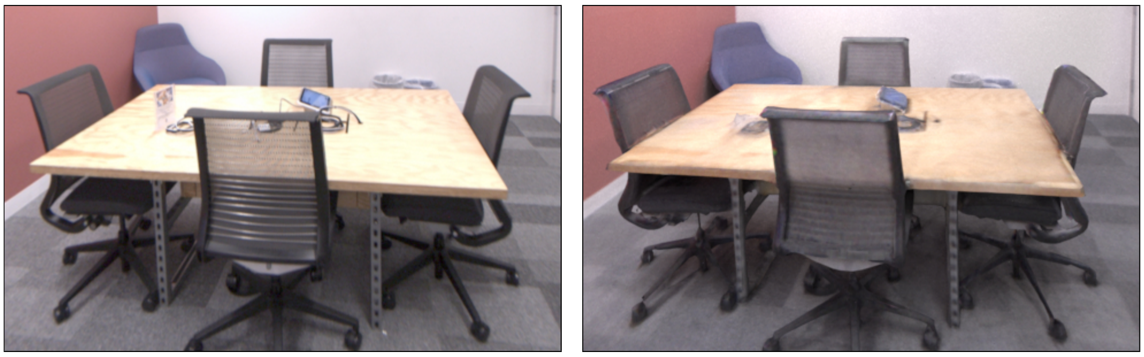


Figure 11.15: Our method enables photorealistic relighting of captured scenes. Here, we simulate lighting variations at three different times of day (**a**-**c**) in the ROOM-0 scene. The scene reacts correctly to new illumination, which differs significantly from the original indoor lighting it was captured with (**d**).



(a) Reference (b) Optimized

Figure 11.16: Since the reflectance model used by our method does not support transparency, the color of the table seen through the back of the chair is incorrectly attributed to the chair itself.

example is shown in Figure 11.16: the back of the chair is incorrectly assigned the color of the table that should have been seen through it. Differentiable rendering is generally well suited to support advanced effects such as transparency and refractions, as the corresponding light transport is well understood. However, we have limited our simulation to the most common and important effects in order to reduce the underlying optimization complexity and improve robustness of the method in common scenarios.

Future work. We believe our method provides an important stepping stone towards full scene understanding, which opens up new opportunities for scenarios where realism is important, such as augmented and mixed reality, robotics and sensory simulation, and synthetic augmentation of datasets.

Despite high-quality input data, the accuracy and robustness of reconstructions can still be improved. Some of the heuristics used in the method were appropriate for the indoor setting, but may not generalize to outdoor or very different scenes. More work is needed on principled priors for reconstruction from real data—perhaps using a data-driven approach.

Our method, being based on a physically based differentiable rendering, could naturally be extended in future work to reconstruct a wider range of appearance (*e.g.* transparent and refractive surfaces), as well as illumination from outdoor scenes (*e.g.* using an environment map). Finally, the imperfections of geometric reconstruction and reference images (such as camera pose, motion blur, sensor noise, etc) could be integrated into the differentiable simulation and minimized to further improve reconstruction [12, 57, 112, 113, 114]—although with each new degree of freedom come new ambiguities and challenging local minima.

12 | Conclusion

Over the last five years, interest in differentiable physically based rendering research has grown significantly, leading to fast-paced progress. Our work contributes to the field on three axes: algorithms, systems, and applications.

Contributions. First off, we proposed algorithms for unbiased and efficient estimation of scene parameter gradients through physically based rendering algorithms. We initially considered the usage of automatic differentiation and examined the associated tradeoffs. AD is a powerful tool for fast experimental development, but comes with significant memory requirements, which limits its applicability. In response, we have developed an adjoint method for differentiable rendering, which recasts the gradient estimation problem as a modified light transport problem. This unlocked vastly more efficient implementations, and opened the door to applying decades of rendering research to the adjoint problem. In the presence of participating media, such as clouds, primal rendering remains challenging and computationally expensive. Volumes are among the phenomena that need physically based algorithms the most in order to be rendered accurately. We identified a flaw in existing differentiable volume rendering approaches, and proposed a dedicated importance sampling technique yielding unbiased and low variance gradients.

Next, together with multiple collaborators at EPFL's Realistic Graphics Lab, we designed and developed systems to support the efficient implementation of these algorithms, as well as effective research. Mitsuba 2 is an open-source retargetable forward and differentiable renderer. It supports multiple representations of light (from monochromatic to spectral and polarized light), computational backends (CPU, GPU), numerical precisions, as well as automatic differentiation. By conducting our research and developing the system in parallel, we learned about the tradeoffs involved and the requirements imposed by future differentiable rendering research. With the radiative backpropagation algorithm, it became possible to contain the usage of AD to smaller components such as BSDF and emitters. In turn, this enabled JIT compilation of the entire rendering algorithm with symbolic execution—AD included. Because the AD graph no longer needs to be maintained at rendering time, the resulting implementation uses orders of magnitude less memory and time. These findings were incorporated, generalized and systematized by Jakob et al. [80] into the Dr.Jit library, which underpins the Mitsuba 3 system [79].

Finally, we brought together our algorithms and systems to tackle challenging inverse rendering scenarios. Our caustic design application demonstrated the power and flexibility of the AD-based approach. A simple implementation of the forward pass, combined with a stochastic gradient descent loop, can compete with hand-derived methods and even generalize to more complex settings such as mixing colored emitters and GRIN lenses. Using our differential ratio tracking sampler, we carried out difficult inverse volume reconstructions, faithfully recovering the density and albedo of participating media. After encountering suboptimal local minima in our experiments, we proposed leveraging a non-physical, NeRF-style representation as initialization to bootstrap the physically-based reconstruction. Moving to real-world data, we recovered the lighting and material properties of captured indoor scenes. The methods had to deal with many imperfections in the input and includes a collection of practical ideas to overcome them. The recovered material parameters are high-resolution fully relightable, editable, and ready for use in standard graphics pipelines.

Impact. We believe that our contributions, together with other recent advances in the field, constitute a solid foundation for differentiable rendering research.

In fact, follow-up work has already been conducted in all areas discussed above. Path replay backpropagation [21] improves on our radiative backpropagation algorithms, reducing its time complexity and generalizing it to specular materials. DR.JIT [80] and Mitsuba 3 build upon Mitsuba 2 and the megakernel conversion technique of Chapter 8 to deliver a fast and flexible platform for differentiable rendering research and applications.

Finally, our algorithms and systems were put to use in a range of articles, from 3D printing [130] to millimeter wave radar simulation [169] and the identification of materials and pose of space debris [135]. A literature review conducted at the time of writing reveals that 72 published works use, build onto or extend Mitsuba 2.

The rapidly growing set of practical applications spans many fields, including light field capture [118], cloud tomography [132, 133, 134] non-line of sight imaging [36, 221], computational lens design [129], as well as automatic simplification and reconstruction of video game assets [222, 223].

Outlook. The research presented in this thesis notably lacked support for derivatives at discontinuities, such as silhouette edges. In practice, this means that the proposed algorithms cannot be used on their own to optimize the shape of objects or their positions. Luckily, dedicated methods are actively researched [57, 112, 113], and can be combined with ours [12]. At the time of writing, the available methods for unbiased gradient estimation at silhouette discontinuities still add significant implementation complexity and runtime. More research is needed for a lightweight and general solution, including support for implicit shape representations [114].

The design of specialized sampling strategies that improve the variance of gradient estimates, akin to Chapter 5, is also a promising avenue for future work. More broadly, a better understanding of the effect of quality tradeoffs is needed to optimally utilize the available compute budgets.

The foundation that has been laid opens the door to a wealth of applications. Indeed, many scientific problems involve an imaging process of some kind—from physics to climate science to optics and medical imaging. Additional research is required to apply differentiable rendering tools effectively to these concrete problems. As we have seen in Chapter 11, real-world data has its own challenges: noisy observations, imperfect calibration and models, etc, will all need to be handled robustly. Perhaps even more importantly, every new application comes with under-determinism, ambiguities, and various non-convexities. We can expect domain-specific priors, modeling assumptions, and initialization schemes to be part of the solution. Increasing the scope of the input data, *e.g.* to include spectral or polarization measurements, can also help remove ambiguities during reconstruction [33]. However, a broadly applicable, general method for robust convergence on entire classes of inverse problems is desirable. Other scientific fields face similar challenges, and it will be important to learn from them as well as share back any advances.

Appendices

A | Inverse Volume Rendering with the Null-Scattering Integral Formulation

In Section 5.3.2, we identified a singularity $1/\sigma_t$ when evaluating in-scattering gradients using free-flight sampling based estimators. We now show that this singularity materializes as well when deriving the estimators starting from the null-scattering integral formulation.

Null-scattering form. The null-scattering integral form of the radiative transfer equation [224, 225, 226] is

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = \int_0^{t_s} \bar{\sigma}\,\bar{\mathrm{T}}(t) \left[\frac{\sigma_a(t)}{\bar{\sigma}} L_e(t) + \frac{\sigma_s(t)}{\bar{\sigma}} L_s(t, \boldsymbol{\omega}) + \frac{\sigma_n(t)}{\bar{\sigma}} L_i(t, \boldsymbol{\omega}) \right] \mathrm{d}t \\ + \mathrm{T}(t_s) \left[L_e(t_s) + L_s(t_s, \boldsymbol{\omega}) \right], \tag{A.1}$$

where t_s is the distance to the next surface along the ray $(\mathbf{x}, \boldsymbol{\omega})$, *e.g.* the medium's boundary. $\bar{\mathrm{T}}(t) = \exp(-\bar{\sigma}\, t)$ corresponds to the transmittance of the homogenized medium and $\sigma_n = \bar{\sigma} - \sigma_t$.

Adjoint null-scattering radiative transfer equation. Assuming $L_e = 0$ within the medium and using the $\boldsymbol{\theta} = (\sigma_t, \alpha)$ parametrization, Equation (A.1) simplifies to

$$L_i(\mathbf{x}, \boldsymbol{\omega}) = \int_0^{t_s} \bar{\sigma}\,\bar{\mathrm{T}}(t) \left[\frac{\sigma_t(t)\,\alpha(t)}{\bar{\sigma}} L_s(t, \boldsymbol{\omega}) + \frac{\sigma_n(t)}{\bar{\sigma}} L_i(t, \boldsymbol{\omega}) \right] \mathrm{d}t \\ + \mathrm{T}(t_s) \left[L_e(t_s) + L_s(t_s, \boldsymbol{\omega}) \right]. \tag{A.2}$$

In the following, we consider $\partial_{\boldsymbol{\theta}} \bar{\sigma} = 0$, as the majorant is constant. Taking the derivative with respect to scene parameters $\boldsymbol{\theta}$ and omitting the dependence on $\boldsymbol{\omega}$, we obtain the following terms. Similar to Equation (5.11), the first term captures how the *in-scattered* radiance can increase due to a local density increase:

$$\partial_{\boldsymbol{\theta}} L_i = \int_0^{t_s} \bar{\sigma}\,\mathrm{T}(t)\, \frac{\partial_{\boldsymbol{\theta}} \left[\sigma_t(t)\,\alpha(t) \right]}{\bar{\sigma}} L_s(t)\, \mathrm{d}t \ \cdots \tag{A.3}$$

Recalling that $\sigma_n = 1 - \sigma_t$, we see that the next expression has a role similar to Equation (5.4)—it accounts for the density increases at positions prior to real interactions:

$$\begin{aligned} \cdots + \int_0^{t_s} \bar{\sigma}\, \bar{\mathrm{T}}(t)\, \frac{\partial_\theta \sigma_n(t)}{\bar{\sigma}}\, L_i(t)\, \mathrm{d}t \\ + T(t_s) \left[\int_0^{t_s} -\partial_\theta \sigma_t(t')\, \mathrm{d}t' \right] \left[L_e(t_s) + L_s(t_s) \right] \cdots \end{aligned} \tag{A.4}$$

The third expression captures changes later along the path, which are weighted by the path throughput:

$$\begin{aligned} \cdots + \int_0^{t_s} \bar{\sigma}\, \bar{\mathrm{T}}(t) \left[\frac{\sigma_t(t)\, \alpha(t)}{\bar{\sigma}}\, \partial_\theta L_s(t) + \frac{\sigma_n(t)}{\bar{\sigma}}\, \partial_\theta L_i(t) \right] \mathrm{d}t \\ + T(t_s) \left[\partial_\theta L_e(t_s) + \partial_\theta L_s(t_s) \right]. \end{aligned} \tag{A.5}$$

Free-flight based estimators with null-scattering. Since the null-scattering formulation implies sampling free-flight distances $t \sim \bar{\sigma}\, \bar{\mathrm{T}}(t)$, it may appear as if the problematic $1/\sigma_t$ term discussed in Section 5.3.2 has been avoided. However, it manifests again at the very next step: once the distance t has been sampled from the homogenized medium, testing whether a *real* or *null* (virtual) particle has been encountered is determined by the probabilities

$$p_{\mathrm{real}}(t) = \frac{\sigma_t(t)}{\bar{\sigma}} \quad \text{and} \quad p_{\mathrm{null}}(t) = \frac{\sigma_n(t)}{\bar{\sigma}}, \tag{A.6}$$

reintroducing the problematic factor. Hence, the resulting estimators of adjoint terms (A.3) and (A.4) suffer from the same issue as before:

$$\langle \partial L_1^{\mathrm{N}} \rangle = \frac{\bar{\sigma}}{\sigma_t(t)} \frac{\partial_\theta \left[\sigma_t(t)\, \alpha(t) \right]}{\bar{\sigma}} L_s(t) = \frac{\partial_\theta \left[\sigma_t(t)\, \alpha(t) \right]}{\sigma_t(t)} L_s(t), \tag{A.7}$$

$$\text{and} \quad \langle \partial L_2^{\mathrm{N}} \rangle = \frac{\partial_\theta \sigma_n(t)}{\sigma_n(t)} L_i(t). \tag{A.8}$$

Symmetric issue in null interaction gradients. While we have focused our efforts on issues due to real interactions where $\sigma_t \approx 0$, Tregan et al. [149] have identified a similar issue for $\sigma_n \approx \bar{\sigma}$, which becomes apparent in Equation (A.8). Depending on the application, selecting a sufficiently large majorant $\bar{\sigma}$ can be a practical way to sidestep the problem. Another simple solution would be to estimate transmittance gradients of Equations (5.8) and (A.4) not at locations $t' \in (0, \min(t, t_s))$ corresponding to null interactions, but rather sampled uniformly at random on the segment. Since the integrands do not include other factors beyond $\partial_\theta \sigma_t$, uniform sampling is appropriate. The estimator would then include a sampling weight equal to the length of the segment. Our method generalizes the solution of Tregan et al. by handling both the in-scattering and transmittance gradient singularities.

In practice, we found transmittance gradients to be well-behaved and did not observe significant differences between these alternatives when using a majorant $1.01\times$ larger than the maximum σ_t value.

B | Inverse Rendering of Real Rooms — additional results

We include additional validations for the reconstruction method of Chapter 11.

B.1 Comparison to prior work

We compare to the method of Azinović et al. [128] for joint materials and lighting estimation in Figure B.1. To this end, we modified the authors' implementation to support Replica's multiplexed HDR captured images. Spatially-varying parameters are supported in their method by subdividing the geometry and assigning one material per triangle. The optimization is run with the settings recommended in the paper for 6 million iterations, on the same input data as ours.

Their method only accounts for the first two bounces of light transport and must thus produce an overly bright base color to fit the reference, which is especially apparent in shadowed regions, where most light comes from indirect reflection. Roughness and specular, as well as other BRDF parameters (not shown), are optimized freely, which leads to implausible high-frequency variations across surfaces. Finally, a significant amount of Monte Carlo noise is present in the optimized spatially-varying parameters.

In contrast, our pipeline allows us to simulate full global illumination with a large number of light bounces (we use 8 in practice as the contribution from further bounces is minimal). Combined with our texture-space sampling method and inductive biases, our method produces plausible and noise-free results with more precisely reconstructed albedo, material parameters, and illumination. Finally, shadows are effectively removed from the albedo texture, and view-dependent effects such as specular highlights are correctly attributed to material parameters.

B.2 Ablation study

In order to evaluate the impact of each of our method's components and design decisions, we have conducted a detailed ablation study. We reconstruct emission and material parameters of the OFFICE-0 scene using 9 variants of our method, progressively adding the features described in Section 11.3. We run each variant for 130 minutes and compute the pixel-wise Mean Squared Relative Error (MSRE) with respect to a fixed set of 30

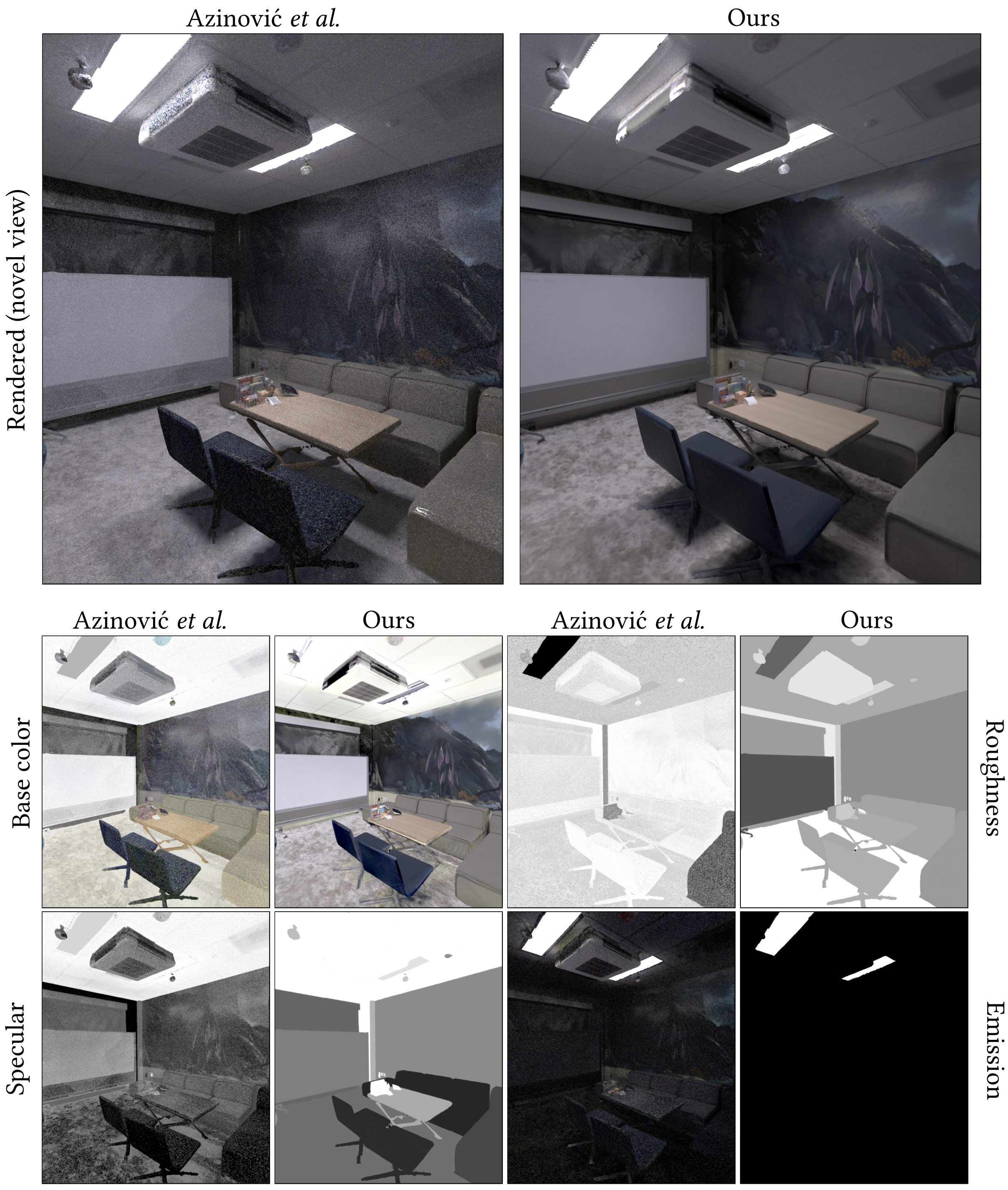


Figure B.1: Given the same inputs, previous work based on differentiable path tracing [128] outputs textures contaminated by Monte Carlo rendering noise and exhibits several of the issues outlined in Section 11.3, including uneven convergence and implausible high-frequency changes in roughness & specular material parameters.

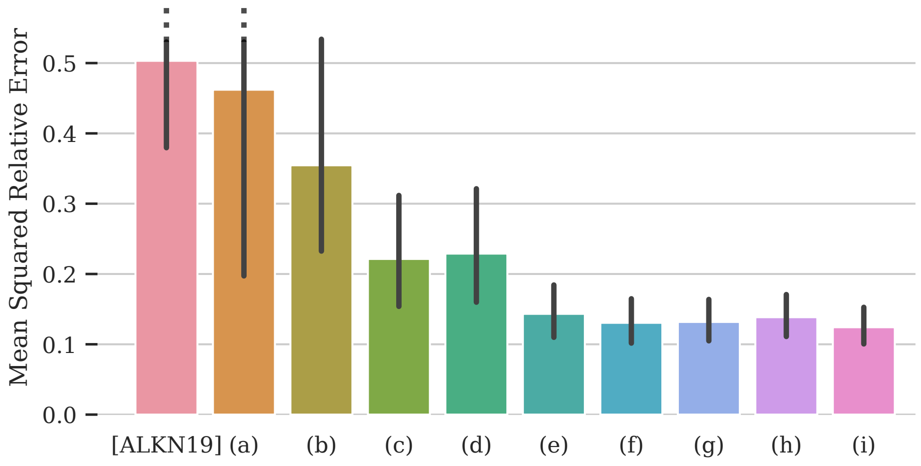


Figure B.2: Ablation study: we progressively add features to a naive inverse rendering baseline **(a)** up to our full method **(i)**. The method of Azinović et al. [128] is included for comparison. Features **(e-i)** result in mostly qualitative improvements, that are visible on a close-up of a wall in Figure B.3.

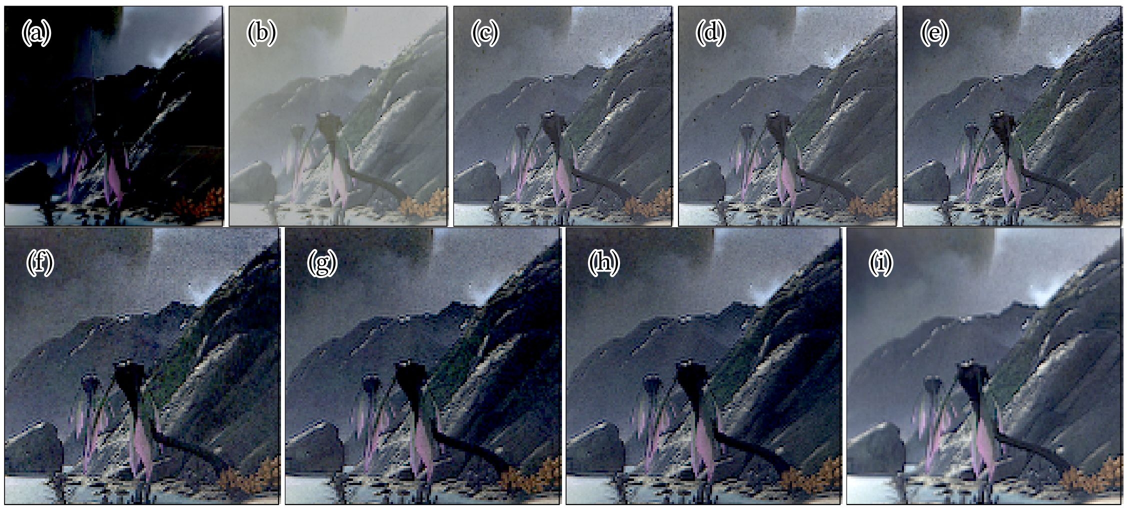


Figure B.3: Detailed crops of the recovered wall of OFFICE-0 at each step of the ablation study of Figure B.2. Our full method **(i)** recovers the most details while avoiding noise in the texture entirely.

reference images chosen at random. The results are shown in Figure B.2. Each feature improves the re-rendering loss and / or helps achieve more plausible results. For visual inspection, a close-up of the recovered wall appearance for each variant is shown in Figure B.3.

The baseline **(a)** uses image-based optimization, the most direct application of differentiable rendering. Variant **(b)** uses our novel texture-space sampling method, described in Section 11.3.3. Variants **(c)** and **(d)** add our parametrizations of emitters and materials respectively (Section 11.3.2). The inductive bias on materials does not decrease error in this experiment, but does ensure more plausible results. Understandably, unconstrained optimization may achieve good error by overfitting, but produce implausible material parameters (see Figures 11.6 and 11.7). Variant **(e)** initializes the diffuse albedo parameter with the median of all observations. Variant **(f)** clamps gradients to prevent Monte Carlo noise from contaminating the textured parameters (Section 11.3.4), while variant **(g)** additionally prevents the Adam state updates and momentum to be applied to variables that were not observed in the current iteration. Variant **(h)** applies Polyak-Ruppert averaging [219, 220]. Finally, variant **(i)** uses a coarse-to-fine scheme, progressively introducing degrees of freedom to the optimization (Section 11.3.4).

References

[1] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2019. URL: https://www.blender.org.

[2] Chris Lattner and Vikram Adve. “LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation”. In: *CGO*. San Jose, CA, USA, Mar. 2004, pp. 75–88.

[3] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[4] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[5] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56 –61. DOI: 10.25080/Majora-92bf1922-00a.

[6] The pandas development team. “pandas-dev/pandas: Pandas”. In: (Oct. 2022). DOI: 10.5281/zenodo.7223478.

[7] Anne-Wil Harzing. *Publish or Perish*. https://harzing.com/resources/publish-or-perish. 2007.

[8] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. *pybind11 – Seamless operability between C++11 and Python*. https://github.com/pybind/pybind11. 2017.

[9] Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Brianna Laugher, and Florian Bruhin. *pytest 7.1*. 2004. URL: https://github.com/pytest-dev/pytest.

[10] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[11] Michael L. Waskom. “seaborn: statistical data visualization”. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[12] Tizian Zeltner, Sébastien Speierer, Iliyan Georgiev, and Wenzel Jakob. “Monte Carlo Estimators for Differential Light Transport”. In: *Transactions on Graphics (Proceedings of SIGGRAPH)* 40.4 (Aug. 2021), pp. 1–16. ISSN: 0730-0301. DOI: 10.1145/3450626.3459807.

[13] Wenzel Jakob. *Enoki: structured vectorization and differentiation on modern processor architectures.* https://github.com/mitsuba-renderer/enoki. 2019.

[14] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. “Mitsuba 2: A Retargetable Forward and Inverse Renderer”. In: *Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38.6 (Dec. 2019), pp. 1–17. ISSN: 0730-0301. DOI: 10.1145/3355089.3356498.

[15] Merlin Nimier-David, Sébastien Speierer, Benoît Ruiz, and Wenzel Jakob. “Radiative Backpropagation: An Adjoint Method for Lightning-Fast Differentiable Rendering”. In: *Transactions on Graphics (Proceedings of SIGGRAPH)* 39.4 (July 2020), p. 146. ISSN: 0730-0301. DOI: 10.1145/3386569.3392406.

[16] Merlin Nimier-David, Zhao Dong, Wenzel Jakob, and Anton Kaplanyan. “Material and Lighting Reconstruction for Complex Indoor Scenes with Texture-space Differentiable Rendering”. In: *Eurographics Symposium on Rendering - DL-only Track*. Ed. by Adrien Bousseau and Morgan McGuire. The Eurographics Association, 2021, pp. 73–84. ISBN: 978-3-03868-157-1. DOI: 10.2312/sr.20211292.

[17] Merlin Nimier-David, Thomas Müller, Alexander Keller, and Wenzel Jakob. “Unbiased Inverse Volume Rendering with Differential Trackers”. In: *ACM Trans. Graph.* 41.4 (July 2022), 44:1–44:20. ISSN: 0730-0301. DOI: 10.1145/3528223.3530073. URL: https://doi.org/10.1145/3528223.3530073.

[18] Russel E. Caflisch. “Monte Carlo and quasi-Monte Carlo methods”. In: *Acta Numerica* 7 (Jan. 1998), 1–49. ISSN: 0962-4929. DOI: 10.1017/S0962492900002804.

[19] Eric Veach and Leonidas J. Guibas. “Optimally Combining Sampling Techniques for Monte Carlo Rendering”. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’95. New York, NY, USA: Association for Computing Machinery, 1995, 419–428. ISBN: 0897917014. DOI: 10.1145/218380.218498. URL: https://doi.org/10.1145/218380.218498.

[20] Ivo Kondapaneni, Petr Vévoda, Pascal Grittmann, Tomáš Skřivan, Philipp Slusallek, and Jaroslav Křivánek. “Optimal multiple importance sampling”. In: *ACM Transactions on Graphics (TOG)* 38.4 (Aug. 2019), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3306346.3323009.

[21] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. “Path Replay Backpropagation: Differentiating Light Paths using Constant Memory and Linear Time”. In: *Transactions on Graphics (Proceedings of SIGGRAPH)* 40.4 (Aug. 2021), 108:1–108:14. ISSN: 0730-0301. DOI: 10.1145/3450626.3459804.

[22] Melissa E. O’Neill. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation.* Tech. rep. HMC-CS-2014-0905. Claremont, CA: Harvey Mudd College, Sept. 2014.

[23] Robert L Cook. “Stochastic sampling in computer graphics”. In: *ACM Transactions on Graphics (TOG)* 5.1 (Jan. 1986), pp. 51–72. ISSN: 0730-0301. DOI: 10.1145/7529.8927.

[24] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Nov. 2016, p. 1266. ISBN: 9780128006450.

[25] Eric Veach. *Robust monte carlo methods for light transport simulation.* 1610. Stanford University PhD thesis, 1997.

[26] Wenzel Jakob. *Mitsuba renderer.* Version 0.6. 2010. URL: http://www.mitsuba-renderer.org.

[27] David L. MacAdam. “Maximum Visual Efficiency of Colored Materials”. In: *J. Opt. Soc. Am.* 25.11 (Nov. 1935), pp. 361–367. ISSN: 0030-3941. DOI: 10.1364/JOSA.25.000361. URL: http://opg.optica.org/abstract.cfm?URI=josa-25-11-361.

[28] Johannes Meng, Florian Simon, Johannes Hanika, and Carsten Dachsbacher. “Physically Meaningful Rendering using Tristimulus Colours”. In: *Computer Graphics Forum* 34.4 (July 2015), pp. 31–40. ISSN: 0167-7055. DOI: https://doi.org/10.1111/cgf.12676. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12676. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12676.

[29] Wenzel Jakob and Johannes Hanika. “A Low-Dimensional Function Space for Efficient Spectral Upsampling”. In: *Computer Graphics Forum (Proceedings of Eurographics)* 38.2 (Mar. 2019), pp. 147–155. ISSN: 0167-7055.

[30] Mengqi (Mandy) Xia, Bruce Walter, Eric Michielssen, David Bindel, and Steve Marschner. “A Wave Optics Based Fiber Scattering Model”. In: *ACM Trans. Graph.* 39.6 (Nov. 2020), pp. 1–16. ISSN: 0730-0301. DOI: 10.1145/3414685.3417841. URL: https://doi.org/10.1145/3414685.3417841.

[31] Seung-Hwan Baek, Tizian Zeltner, Hyun Jin Ku, Inseung Hwang, Xin Tong, Wenzel Jakob, and Min H. Kim. “Image-Based Acquisition and Modeling of Polarimetric Reflectance”. In: *ACM Trans. Graph.* 39.4 (July 2020), p. 139. ISSN: 0730-0301. DOI: 10.1145/3386569.3392387. URL: https://doi.org/10.1145/3386569.3392387.

[32] Inseung Hwang, Daniel S. Jeon, Adolfo Muñoz, Diego Gutierrez, Xin Tong, and Min H. Kim. “Sparse Ellipsometry: Portable Acquisition of Polarimetric SVBRDF and Shape with Unstructured Flash Photography”. In: *ACM Trans. Graph.* 41.4 (July 2022), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3528223.3530075. URL: https://doi.org/10.1145/3528223.3530075.

[33] Vincent Bosboom, Matthias Schlottbom, and Felix Schwenninger. “On the unique solvability of radiative transfer equations with polarization”. In: *arXiv preprint arXiv:2203.03233* (Mar. 2022).

[34] James T Kajiya. “The rendering equation”. In: *ACM Siggraph Computer Graphics.* Ed. by David C. Evans and Russell J. Athay. Vol. 20. 4. ACM. ACM, July 1986, pp. 157–164. DOI: 10.1145/280811.280987.

[35] Subrahmanyan Chandrasekhar. *Radiative transfer.* New York: Dover publications, Mar. 1960, pp. 27–57.

[36] Chia-Yin Tsai, Aswin C Sankaranarayanan, and Ioannis Gkioulekas. "Beyond Volumetric Albedo–A Surface Optimization Framework for Non-Line-Of-Sight Imaging". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* IEEE, June 2019, pp. 1545–1555. DOI: 10.1109/cvpr.2019.00164. URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Tsai_Beyond_Volumetric_Albedo_--_A_Surface_Optimization_Framework_for_Non-Line-Of-Sight_CVPR_2019_paper.html.

[37] Kenneth M Case. *Linear transport theory.* Addison-Wesley Publishing Company, Oct. 1967. DOI: 10.1063/1.3034554.

[38] James T. Kajiya and Brian P Von Herzen. "Ray Tracing Volume Densities". In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984). Ed. by Hank Christiansen, 165–174. ISSN: 0097-8930. DOI: 10.1145/964965.808594. URL: https://doi.org/10.1145/964965.808594.

[39] Jenni Heino, Simon Arridge, Jan Sikora, and Erkki Somersalo. "Anisotropic effects in highly scattering media". In: *Phys. Rev. E* 68 (3 Sept. 2003), p. 031908. ISSN: 1063-651X. DOI: 10.1103/PhysRevE.68.031908. URL: https://link.aps.org/doi/10.1103/PhysRevE.68.031908.

[40] A Kienle, FK Forster, and R Hibst. "Anisotropy of light propagation in biological tissue". In: *Optics letters* 29.22 (Nov. 2004), pp. 2617–2619. ISSN: 0146-9592.

[41] Juha Heiskala, Ilkka Nissilä, Tuomas Neuvonen, Seppo Järvenpää, and Erkki Somersalo. "Modeling anisotropic light propagation in a realistic model of the human head". In: *Appl. Opt.* 44.11 (Apr. 2005), pp. 2049–2057. ISSN: 0003-6935. DOI: 10.1364/AO.44.002049. URL: http://opg.optica.org/ao/abstract.cfm?URI=ao-44-11-2049.

[42] Wenzel Jakob, Jonathan T. Moon, Adam Arbree, Kavita Bala, and Steve Marschner. "A Radiative Transfer Framework for Rendering Materials with Anisotropic Structure". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 29.10 (July 2010), 53:1–53:13. ISSN: 0730-0301. DOI: 10.1145/1778765.1778790. URL: http://www.cs.cornell.edu/projects/diffusion-sg10/diffusion-sg10-tr.pdf.

[43] Alexander B. Kostinski. "On the extinction of radiation by a homogeneous but spatially correlated random medium". In: *J. Opt. Soc. Am. A* 18.8 (Aug. 2001), pp. 1929–1933. ISSN: 1084-7529. DOI: 10.1364/JOSAA.18.001929. URL: http://opg.optica.org/josaa/abstract.cfm?URI=josaa-18-8-1929.

[44] Edward W. Larsen and Richard Vasques. “A generalized linear Boltzmann equation for non-classical particle transport”. In: *Journal of Quantitative Spectroscopy and Radiative Transfer* 112.4 (Mar. 2011). 2009 International Conference on Mathematics and Computational Methods (M&C 2009), pp. 619–631. ISSN: 0022-4073. DOI: https://doi.org/10.1016/j.jqsrt.2010.07.003. URL: https://www.sciencedirect.com/science/article/pii/S0022407310002827.

[45] Adrian Jarabo, Carlos Aliaga, and Diego Gutierrez. “A Radiative Transfer Framework for Spatially-Correlated Materials”. In: *ACM Trans. Graph.* 37.4 (July 2018), pp. 1–13. ISSN: 0730-0301. DOI: 10.1145/3197517.3201282. URL: https://doi.org/10.1145/3197517.3201282.

[46] Benedikt Bitterli, Srinath Ravichandran, Thomas Müller, Magnus Wrenninge, Jan Novák, Steve Marschner, and Wojciech Jarosz. “A radiative transfer framework for non-exponential media”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 37.6 (Nov. 2018), 225:1–225:17. ISSN: 0730-0301. DOI: 10/gfz2cm.

[47] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. “Monte Carlo Techniques for Direct Lighting Calculations”. In: *ACM Trans. Graph.* 15.1 (Jan. 1996), 1–36. ISSN: 0730-0301. DOI: 10.1145/226150.226151. URL: https://doi.org/10.1145/226150.226151.

[48] Pat Hanrahan and Wolfgang Krueger. “Reflection from Layered Surfaces Due to Subsurface Scattering”. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’93. Anaheim, CA: Association for Computing Machinery, 1993, 165–174. ISBN: 0897916018. DOI: 10.1145/166117.166139. URL: https://doi.org/10.1145/166117.166139.

[49] Eric Veach and Leonidas Guibas. “Bidirectional estimators for light transport”. In: *Photorealistic Rendering Techniques.* Springer, 1995, pp. 145–167. ISBN: 9783642878275. DOI: 10.1007/978-3-642-87825-1_11. URL: http://www.cs.jhu.edu/~misha/ReadingSeminar/Papers/Veach94.pdf.

[50] Henrik Wann Jensen. *Realistic image synthesis using photon mapping.* Vol. 364. Ak Peters Natick, July 2001. ISBN: 9781568811475. DOI: 10.1201/b10685.

[51] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. “Progressive Photon Mapping”. In: *ACM SIGGRAPH Asia 2008 Papers.* Vol. 27. SIGGRAPH Asia ’08. Singapore: Association for Computing Machinery, 2008, p. 130. ISBN: 9781450318310. DOI: 10.1145/1457515.1409083. URL: https://doi.org/10.1145/1457515.1409083.

[52] Alexander Keller. “Instant Radiosity”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’97. USA: ACM Press/Addison-Wesley Publishing Co., 1997, 49–56. ISBN: 0897918967. DOI: 10.1145/258734.258769. URL: https://doi.org/10.1145/258734.258769.

[53] Wenzel Jakob. “Light Transport On Path-Space Manifolds”. PhD thesis. Cornell University, 2013.

[54] Johannes Hanika, Marc Droske, and Luca Fascione. “Manifold Next Event Estimation”. In: *Computer Graphics Forum* 34.4 (2015), pp. 87–97. DOI: https://doi.org/10.1111/cgf.12681. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12681. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12681.

[55] Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. “Specular Manifold Sampling for Rendering High-Frequency Caustics and Glints”. In: *Transactions on Graphics (Proceedings of SIGGRAPH)* 39.4 (July 2020), p. 149. ISSN: 0730-0301. DOI: 10.1145/3386569.3392408.

[56] Adam Smith James Skorupski James Davis. “Transient Rendering”. In: (2008).

[57] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. “Differentiable Monte Carlo Ray Tracing Through Edge Sampling”. In: *ACM Transactions on Graphics* 37.6 (Dec. 2018), pp. 1–11. ISSN: 0730-0301. DOI: 10.1145/3272127.3275109. URL: https://dl.acm.org/doi/pdf/10.1145/3272127.3275109.

[58] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation.* Vol. 105. SIAM, 2008, pp. I–XXIV, 1–369.

[59] R. E. Wengert. “A Simple Automatic Derivative Evaluation Program”. In: *Commun. ACM* 7.8 (Aug. 1964), 463–464. ISSN: 0001-0782. DOI: 10.1145/355586.364791. URL: https://doi.org/10.1145/355586.364791.

[60] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1”. In: Cambridge, MA, USA: MIT Press, 1986. Chap. Learning Internal Representations by Error Propagation.

[61] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y.Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K.

Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* 2015.

[62] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic differentiation in PyTorch". In: (2017).

[63] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. *JAX: composable transformations of Python+NumPy programs.* Version 0.3.13. 2018. URL: `http://github.com/google/jax`.

[64] Yu M Volin and GM Ostrovskii. "Automatic computation of derivatives with the use of the multilevel differentiating technique—1. Algorithmic basis". In: *Computers & mathematics with applications* 11.11 (Nov. 1985), pp. 1099–1114. ISSN: 0898-1221. DOI: `10.1016/0898-1221(85)90188-9`.

[65] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes.* CRC Press, May 1962. ISBN: 9780203749319. DOI: `10.1201/9780203749319`.

[66] Joel Andersson. "A general-purpose software framework for dynamic optimization". PhD thesis. KU Leuven, 2013.

[67] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. "Fluid control using the adjoint method". In: *ACM Transactions On Graphics (TOG).* Vol. 23. 3. ACM. ACM Press, Aug. 2004, pp. 449–456. DOI: `10.1145/1186562.1015744`.

[68] Shayan Hoshyari, Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. "Vibration-minimizing motion retargeting for robotic characters". In: *ACM Transactions on Graphics (TOG)* 38.4 (Aug. 2019), pp. 1–14. ISSN: 0730-0301. DOI: `10.1145/3306346.3323034`.

[69] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural ordinary differential equations". In: *Advances in neural information processing systems.* Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. June 2018, pp. 6571–6583. URL: `https://dl.acm.org/doi/10.5555/3327757.3327764`.

[70] Andrew M Bradley. *PDE-constrained optimization and the adjoint method.* Tech. rep. Technical Report. Stanford University., 2013. URL: `https://cs.stanford.edu/~ambrad/adjoint_tutorial.pdf`.

[71] Matt Pharr and William R Mark. “ispc: A SPMD compiler for high-performance CPU programming”. In: *2012 Innovative Parallel Computing (InPar)*. IEEE. IEEE, May 2012. DOI: `10.1109/inpar.2012.6339601`.

[72] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. “Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines”. In: *SIGPLAN Notices* 48.6 (June 2013). Ed. by Hans-Juergen Boehm and Cormac Flanagan, pp. 519–530. DOI: `10.1145/2491956.2462176`.

[73] Tzu-Mao Li, Michaël Gharbi, Andrew Adams, Frédo Durand, and Jonathan Ragan-Kelley. “Differentiable programming for image processing and deep learning in Halide”. In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 37.4 (Aug. 2018), pp. 1–13. ISSN: 0730-0301. DOI: `10.1145/3197517.3201383`. URL: `https://dl.acm.org/doi/pdf/10.1145/3197517.3201383`.

[74] Luke Anderson, Tzu-Mao Li, Jaakko Lehtinen, and Frédo Durand. “Aether: An embedded domain specific sampling language for Monte Carlo rendering”. In: *ACM Transactions on Graphics* 36.4 (2017), 99:1–99:16. DOI: `10.1145/3072959.3073704`.

[75] Arsène Pérard-Gayot, Richard Membarth, Roland Leißa, Sebastian Hack, and Philipp Slusallek. “Rodent: Generating Renderers without Writing a Generator”. In: *ACM Transactions on Graphics* 38.4 (July 2019), 40:1–40:12. DOI: `10.1145/3306346.3322955`.

[76] Roland Leißa, Klaas Boesche, Sebastian Hack, Arsène Pérard-Gayot, Richard Membarth, Philipp Slusallek, André Müller, and Bertil Schmidt. “AnyDSL: A Partial Evaluation Framework for Programming High-Performance Libraries”. In: *Proceedings of the ACM on Programming Languages (PACMPL)* 2.OOPSLA (Nov. 2018), pp. 1–30. ISSN: 2475-1421. DOI: `10.1145/3276489`. URL: `https://doi.org/10.1145/3276489`.

[77] Yuting Yang, Connelly Barnes, Andrew Adams, and Adam Finkelstein. “Aδ: Autodiff for Discontinuous Programs - Applied to Shaders”. In: *SIGGRAPH, to appear.* Aug. 2022.

[78] Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. “Systematically differentiating parametric discontinuities”. In: *ACM Transactions on Graphics (TOG)* 40.4 (Aug. 2021), pp. 1–18. ISSN:

0730-0301. DOI: 10.1145/3476576.3476671. URL: https://dl.acm.org/doi/pdf/10.1145/3450626.3459775.

[79] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. *Mitsuba 3 renderer*. Version 3.0.0. 2022. URL: https://mitsuba-renderer.org.

[80] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. "DR.JIT: A Just-in-Time Compiler for Differentiable Rendering". In: *ACM Trans. Graph.* 41.4 (July 2022), pp. 1–19. ISSN: 0730-0301. DOI: 10.1145/3528223.3530099. URL: https://doi.org/10.1145/3528223.3530099.

[81] Wenzel Jakob and Steve Marschner. "Manifold exploration: a Markov Chain Monte Carlo technique for rendering scenes with difficult specular transport". In: *ACM Transactions on Graphics* 31.4 (2012), 58:1–58:13. DOI: 10.1145/2185520.2185554.

[82] Tzu-Mao Li, Jaakko Lehtinen, Ravi Ramamoorthi, Wenzel Jakob, and Frédo Durand. "Anisotropic gaussian mutations for metropolis light transport through hessian-hamiltonian dynamics". In: *ACM Transactions on Graphics* 34.6 (Nov. 2015), pp. 1–13. ISSN: 0730-0301. DOI: 10.1145/2816795.2818084.

[83] H Rief, EM Gelbard, RW Schaefer, and KS Smith. "Review of Monte Carlo techniques for analyzing reactor perturbations". In: *Nuclear Science and Engineering* 92.2 (Feb. 1986), pp. 289–297. ISSN: 0029-5639. DOI: 10.13182/nse86-a18178.

[84] Iván Lux and László Koblinger. *Monte Carlo particle transport methods: neutron and photon calculations*. CRC press, May 1991. ISBN: 9781351074834. DOI: 10.1201/9781351074834.

[85] M. C. G. Hall. "Cross-Section Adjustment with Monte Carlo Sensitivities: Application to the Winfrith Iron Benchmark". In: *Nuclear Science and Engineering* 81.3 (1982), pp. 423–431. DOI: 10.13182/NSE82-A20283. eprint: https://doi.org/10.13182/NSE82-A20283. URL: https://doi.org/10.13182/NSE82-A20283.

[86] Matthew M. Loper and Michael J. Black. "OpenDR: An Approximate Differentiable Renderer". In: *ECCV*. Ed. by David J. Fleet, Tomás Pajdla, Bernt Schiele, and Tinne Tuytelaars. Vol. 8695. Springer International Publishing, Sept. 2014, pp. 154–169. ISBN: 9783319105833. DOI: 10.1007/978-3-319-10584-0_11. URL: http://files.is.tue.mpg.de/black/papers/OpenDR.pdf.

[87] Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. “A Versatile Scene Model with Differentiable Visibility Applied to Generative Pose Estimation”. In: *Proceedings of ICCV 2015*. Vol. abs/1602.03725. IEEE, Dec. 2015. DOI: 10.1109/iccv.2015.94. URL: http://arxiv.org/abs/1602.03725.

[88] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. “Neural 3d mesh renderer”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE, June 2018, pp. 3907–3916. DOI: 10.1109/cvpr.2018.00411. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Kato_Neural_3D_Mesh_CVPR_2018_paper.html.

[89] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. “Soft Rasterizer: Differentiable Rendering for Unsupervised Single-View Mesh Reconstruction”. In: *CoRR* abs/1901.05567 (2019). arXiv: 1901.05567. URL: http://arxiv.org/abs/1901.05567.

[90] Felix Petersen, Amit H. Bermano, Oliver Deussen, and Daniel Cohen-Or. “Pix2Vex: Image-to-Geometry Reconstruction using a Smooth Differentiable Renderer”. In: *CoRR* abs/1903.11149 (2019). arXiv: 1903.11149. URL: http://arxiv.org/abs/1903.11149.

[91] Felix Petersen, Bastian Goldluecke, Christian Borgelt, and Oliver Deussen. “GenDR: A Generalized Differentiable Renderer”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. abs/2204.13845. June 2022, pp. 4002–4011. DOI: 10.48550/arxiv.2204.13845. URL: https://doi.org/10.48550/arXiv.2204.13845.

[92] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. “Modular primitives for high-performance differentiable rendering”. In: *ACM Transactions on Graphics (TOG)* 39.6 (Dec. 2020), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3414685.3417861. URL: https://research.aalto.fi/files/74913509/SCI_Laine2020diffrast_paper.pdf.

[93] Yang Zhou, Lifan Wu, Ravi Ramamoorthi, and Ling-Qi Yan. “Vectorization for Fast, Analytic, and Differentiable Visibility”. In: *ACM Transactions on Graphics (TOG)* 40.3 (July 2021), pp. 1–21. ISSN: 0730-0301. DOI: 10.1145/3452097. URL: https://dl.acm.org/doi/pdf/10.1145/3452097.

[94] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. "Differentiable Vector Graphics Rasterization for Editing and Learning". In: *ACM Trans. Graph.* 39.6 (Nov. 2020), pp. 1–15. ISSN: 0730-0301. DOI: 10.1145/3414685.3417871. URL: https://doi.org/10.1145/3414685.3417871.

[95] Volker Blanz and Thomas Vetter. "A morphable model for the synthesis of 3D faces". In: *SIGGRAPH.* ACM Press, 1999, pp. 187–194. DOI: 10.1145/311535.311556. URL: https://pure.mpg.de/pubman/item/item_1793809_3/component/file_3271006/SIGGRAPH-1999-Blanz.pdf.

[96] Ravi Ramamoorthi and Pat Hanrahan. "A Signal-Processing Framework for Inverse Rendering". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, 2001, 117–128. ISBN: 158113374X. DOI: 10.1145/383259.383271. URL: https://doi.org/10.1145/383259.383271.

[97] Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. "Matching Real Fabrics with Micro-Appearance Models". In: *ACM Trans. Graph.* 35.1 (Dec. 2016), pp. 1–26. ISSN: 0730-0301. DOI: 10.1145/2818648. URL: https://doi.org/10.1145/2818648.

[98] Shuang Zhao, Fujun Luan, and Kavita Bala. "Fitting procedural yarn models for realistic cloth rendering". In: *ACM Trans. Graph.* 35.4 (July 2016), pp. 1–11. ISSN: 0730-0301. DOI: 10.1145/2897824.2925932. URL: https://escholarship.org/content/qt2fw2w3gs/qt2fw2w3gs.pdf?t=okn0gj.

[99] Ioannis Gkioulekas, Anat Levin, and Todd Zickler. "An evaluation of computational imaging techniques for heterogeneous inverse scattering". In: *European Conference on Computer Vision.* Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Vol. 9907. Springer. Springer International Publishing, 2016, pp. 685–701. ISBN: 9783319464862. DOI: 10.1007/978-3-319-46487-9_42. URL: https://zenodo.org/record/4292250/files/inverse.pdf.

[100] Shuang Zhao, Lifan Wu, Frédo Durand, and Ravi Ramamoorthi. "Downsampling Scattering Parameters for Rendering Anisotropic Media". In: *ACM Trans. Graph.* 35.6 (Nov. 2016), pp. 1–11. ISSN: 0730-0301. DOI: 10.1145/2980179.2980228. URL: https://doi.org/10.1145/2980179.2980228.

[101] Chengqian Che, Fujun Luan, Shuang Zhao, Kavita Bala, and Ioannis Gkioulekas. "Inverse Transport Networks". In: *arXiv preprint arXiv:1809.10820* abs/1809.10820 (Sept. 2018). URL: http://arxiv.org/abs/1809.10820.

[102] Zdravko Velinov, Marios Papas, Derek Bradley, Paulo Gotardo, Parsa Mirdehghan, Steve Marschner, Jan Novák, and Thabo Beeler. “Appearance Capture and Modeling of Human Teeth”. In: *ACM Transactions on Graphics* 37.6 (Dec. 2018), pp. 1–13. ISSN: 0730-0301. DOI: 10.1145/3272127.3275098.

[103] Shuang Zhao, Ravi Ramamoorthi, and Kavita Bala. “High-order similarity relations in radiative transfer”. In: *ACM Transactions on Graphics (TOG)* 33.4 (July 2014), pp. 1–12. ISSN: 0730-0301. DOI: 10.1145/2601097.2601104. URL: http://shuangz.com/projects/similarity-sg14/similarity-sg14.pdf.

[104] Mike Kasper, Nima Keivan, Gabe Sibley, and Christoffer R. Heckman. “Light Source Estimation with Analytical Path-tracing”. In: *CoRR* abs/1701.04101 (2017). URL: http://arxiv.org/abs/1701.04101.

[105] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. “Inverse global illumination: Recovering reflectance models of real scenes from photographs”. In: *SIGGRAPH*. 1999, pp. 215–224. DOI: 10.1145/311535.311559.

[106] Edward Zhang, Michael F Cohen, and Brian Curless. “Emptying, refurnishing, and relighting indoor spaces”. In: *ACM Trans. Graph.* 35.6 (Nov. 2016), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/2980179.2982432. URL: http://dl.acm.org/ft_gateway.cfm?id=2982432&type=pdf.

[107] Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. “A differential theory of radiative transfer”. In: *ACM Transactions on Graphics (TOG)* 38.6 (Dec. 2019), pp. 1–16. ISSN: 0730-0301. DOI: 10.1145/3355089.3356522.

[108] Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. “Path-Space Differentiable Rendering”. In: *ACM Trans. Graph.* 39.4 (Aug. 2020), 143:1–143:19. ISSN: 0730-0301. DOI: 10.1145/3386569.3392383.

[109] Cheng Zhang, Zihan Yu, and Shuang Zhao. “Path-Space Differentiable Rendering of Participating Media”. In: *ACM Trans. Graph.* 40.4 (Aug. 2021), 76:1–76:15. ISSN: 0730-0301. DOI: 10.1145/3476576.3476631. URL: https://dl.acm.org/doi/pdf/10.1145/3450626.3459782.

[110] Osborne Reynolds. *Papers on mechanical and physical subjects.* Vol. 3. The University Press, 1903.

[111] Kai Yan, Christoph Lassner, Brian Budge, Zhao Dong, and Shuang Zhao. "Efficient Estimation of Boundary Integrals for Path-Space Differentiable Rendering". In: *ACM Trans. Graph.* 41.4 (July 2022), 123:1–123:13. ISSN: 0730-0301. DOI: 10.1145/3528223.3530080.

[112] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. "Reparameterizing Discontinuous Integrands for Differentiable Rendering". In: *ACM Transactions on Graphics* 38.6 (Dec. 2019), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3355089.3356510. URL: https://hal.inria.fr/hal-02497191/file/differentiable-pt-cov.pdf.

[113] Sai Bangaru, Tzu-Mao Li, and Frédo Durand. "Unbiased Warped-Area Sampling for Differentiable Rendering". In: *ACM Trans. Graph.* 39.6 (Dec. 2020), 245:1–245:18. ISSN: 0730-0301. DOI: 10.1145/3414685.3417833. URL: https://dl.acm.org/doi/pdf/10.1145/3414685.3417833.

[114] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. "Differentiable Signed Distance Function Rendering". In: *Transactions on Graphics (Proceedings of SIGGRAPH)* 41.4 (July 2022), 125:1–125:18. ISSN: 0730-0301. DOI: 10.1145/3528223.3530139.

[115] Cheng Zhang, Zhao Dong, Michael Doggett, and Shuang Zhao. "Antithetic sampling for Monte Carlo differentiable rendering". In: *ACM Transactions on Graphics (TOG)* 40.4 (Aug. 2021), pp. 1–12. ISSN: 0730-0301. DOI: 10.1145/3476576.3476632. URL: https://dl.acm.org/doi/pdf/10.1145/3450626.3459783.

[116] Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. "Real-Time Neural Radiance Caching for Path Tracing". In: *ACM Trans. Graph.* 40.4 (July 2021), pp. 1–16. ISSN: 0730-0301. DOI: 10.1145/3450626.3459812. URL: https://doi.org/10.1145/3450626.3459812.

[117] Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. "Material Editing Using a Physically Based Rendering Network". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017, pp. 2280–2288. DOI: 10.1109/iccv.2017.248. URL: http://arxiv.org/pdf/1708.00106.

[118] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV*. Vol. 65. Springer International Publishing, Mar. 2020, pp. 405–421. ISBN: 9783030584511. DOI: 10.1007/978-3-030-58452-8_24. URL: http://arxiv.org/pdf/2003.08934.

[119] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. “Plenoxels: Radiance Fields without Neural Networks”. In: *arXiv:2112.05131* abs/2112.05131 (Dec. 2021). URL: https://arxiv.org/abs/2112.05131.

[120] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy J. Mitra. “ReLU Fields: The Little Non-linearity That Could”. In: *Transactions on Graphics (Proceedings of SIGGRAPH)* 41.4 (July 2022). Ed. by Munkhtsetseg Nandigjav, Niloy J. Mitra, and Aaron Hertzmann, 13:1–13:8. DOI: 10.1145/3528233.3530707. URL: https://dl.acm.org/doi/pdf/10.1145/3528233.3530707.

[121] Ioannis Gkioulekas, Shuang Zhao, Kavita Bala, Todd Zickler, and Anat Levin. “Inverse Volume Rendering with Material Dictionaries”. In: *ACM Transactions on Graphics* 32.6 (Nov. 2013), pp. 1–13. ISSN: 0730-0301. DOI: 10.1145/2508363.2508377.

[122] Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. “Single-image SVBRDF Capture with a Rendering-aware Deep Network”. In: *ACM Trans. Graph.* 37.4 (Aug. 2018), 128:1–128:15. ISSN: 0730-0301. DOI: 10.1145/3197517.3201378. URL: https://dl.acm.org/doi/pdf/10.1145/3197517.3201378.

[123] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. “Match: differentiable material graphs for procedural material capture”. In: 39 (2020), 196:1–196:15. DOI: 10.1145/3414685.3417781.

[124] Paul Guerrero, Miloš Hašan, Kalyan Sunkavalli, Radomír Měch, Tamy Boubekeur, and Niloy J. Mitra. “MatFormer: A Generative Model for Procedural Materials”. In: *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530173. URL: https://doi.org/10.1145/3528223.3530173.

[125] Yiwei Hu, Chengan He, Valentin Deschaintre, Julie Dorsey, and Holly Rushmeier. “An Inverse Procedural Modeling Pipeline for SVBRDF Maps”. In: *ACM Trans. Graph.* 41.2 (Jan. 2022), pp. 1–17. ISSN: 0730-0301. DOI: 10.1145/3502431. URL: https://doi.org/10.1145/3502431.

[126] Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. “Node Graph Optimization Using Differentiable Proxies”. In: *ACM SIGGRAPH 2022 Conference Proceedings.* Ed. by Munkhtsetseg Nandigjav, Niloy J. Mitra, and Aaron Hertzmann. SIGGRAPH ’22. Vancouver, BC, Canada: Association for Com-

puting Machinery, Aug. 2022, 5:1–5:9. ISBN: 9781450393379. DOI: 10.1145/3528233.3530733. URL: https://doi.org/10.1145/3528233.3530733.

[127] Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. "Unified Shape and SVBRDF Recovery using Differentiable Monte Carlo Rendering". In: *arXiv preprint arXiv:2103.15208* 40 (July 2021), pp. 101–113. ISSN: 0167-7055. URL: http://arxiv.org/pdf/2103.15208.

[128] Dejan Azinović, Tzu-Mao Li, Anton Kaplanyan, and Matthias Nießner. "Inverse Path Tracing for Joint Material and Lighting Estimation". In: *Proceedings of Computer Vision and Pattern Recognition (CVPR), IEEE.* IEEE, June 2019, pp. 2447–2456. DOI: 10.1109/cvpr.2019.00255. URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Azinovic_Inverse_Path_Tracing_for_Joint_Material_and_Lighting_Estimation_CVPR_2019_paper.html.

[129] Qilin Sun, Congli Wang, Fu Qiang, Dun Xiong, and Heidrich Wolfgang. "End-to-end complex lens design with differentiable ray tracing". In: *ACM Transactions on Graphics* 40.4 (2021), pp. 1–13.

[130] Thomas Klaus Nindel, Tomáš Iser, Tobias Rittig, Alexander Wilkie, and Jaroslav Křivánek. "A Gradient-Based Framework for 3D Print Appearance Optimization". In: *ACM Trans. Graph.* 40.4 (July 2021), pp. 1–15. ISSN: 0730-0301. DOI: 10.1145/3450626.3459844. URL: https://doi.org/10.1145/3450626.3459844.

[131] Adam Geva, Yoav Y Schechner, Yonatan Chernyak, and Rajiv Gupta. "X-ray computed tomography through scatter". In: *Proceedings of The European Conference on Computer Vision (ECCV).* Ed. by Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss. Vol. 11218. Springer International Publishing, 2018, pp. 34–50. ISBN: 9783030012632. DOI: 10.1007/978-3-030-01264-9_3.

[132] Yael Sde-Chen, Yoav Y. Schechner, Vadim Holodovsky, and Eshkol Eytan. "3DeepCT: Learning volumetric scattering tomography of clouds". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* IEEE, Oct. 2021, pp. 5671–5682. DOI: 10.1109/iccv48922.2021.00562.

[133] Roi Ronen, Yoav Y. Schechner, and Eshkol Eytan. "4D Cloud Scattering Tomography". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision.* IEEE, Oct. 2021, pp. 5520–5529. DOI: 10.1109/iccv48922.2021.00547.

[134] Tamar Loeub, Aviad Levis, Vadim Holodovsky, and Yoav Y Schechner. “Monotonicity prior for cloud tomography”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16.* Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Vol. 12363. Springer. Springer International Publishing, 2020, pp. 283–299. ISBN: 9783030585228. DOI: 10.1007/978-3-030-58523-5_17.

[135] Max Nussbaum, Ewan Schafer, Zizung Yoon, Denise Keil, and Enrico Stoll. “Spectral Light Curve Simulation for Parameter Estimation from Space Debris”. In: *Aerospace* 9.8 (July 2022), p. 403. ISSN: 2226-4310. DOI: 10.3390/aerospace9080403. URL: https://www.mdpi.com/2226-4310/9/8/403.

[136] Ravi Ramamoorthi, Dhruv Mahajan, and Peter Belhumeur. “A first-order analysis of lighting, shading, and shadows”. In: *ACM Transactions on Graphics (TOG)* 26.1 (Jan. 2007), p. 2. ISSN: 0730-0301. DOI: 10.1145/1189762.1189764. URL: https://academiccommons.columbia.edu/doi/10.7916/D8639XJP/download.

[137] Kevin G Jamieson, Robert Nowak, and Ben Recht. “Query Complexity of Derivative-Free Optimization”. In: *Advances in Neural Information Processing Systems 25.* Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. abs/1209.2434. Curran Associates, Inc., 2012, pp. 2672–2680. URL: http://arxiv.org/abs/1209.2434.

[138] James Arvo. “Analytic methods for simulated light transport”. PhD thesis. Yale University, 1995.

[139] Thomas Müller, Markus Gross, and Jan Novák. “Practical Path Guiding for Efficient Light-Transport Simulation”. In: *Computer Graphics Forum* 36.4 (June 2017), pp. 91–100. ISSN: 0167-7055.

[140] Lukas Balles and Philipp Hennig. “Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 404–413. URL: http://proceedings.mlr.press/v80/balles18a.html.

[141] Alain Petrowski, Gerard Dreyfus, and Claude Girault. “Performance analysis of a pipelined backpropagation parallel algorithm”. In: *IEEE Transactions on Neural Networks* 4.6 (1993), pp. 970–981. ISSN: 1045-9227. DOI: 10.1109/72.286892.

[142] Oskar Elek, Denis Sumin, Ran Zhang, Tim Weyrich, Karol Myszkowski, Bernd Bickel, Alexander Wilkie, and Jaroslav Křivánek. “Scattering-aware Texture Reproduction for 3D Printing”. In: *ACM Transactions on Graphics* 36.6 (Nov. 2017), pp. 1–15. ISSN: 0730-0301. DOI: `10.1145/3130800.3130890`.

[143] Denis Sumin, Tobias Rittig, Vahid Babaei, Thomas Nindel, Alexander Wilkie, Piotr Didyk, Bernd Bickel, Jaroslav Křivánek, Karol Myszkowski, and Tim Weyrich. “Geometry-Aware Scattering Compensation for 3D Printing”. In: *ACM Transactions on Graphics* 38 (Aug. 2019), pp. 1–14. ISSN: 0730-0301. DOI: `10.1145/3306346.3322992`.

[144] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. “NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis”. In: *CVPR*. IEEE, June 2021, pp. 7495–7504. DOI: `10.1109/cvpr46437.2021.00741`. URL: `https://openaccess.thecvf.com/content/CVPR2021/html/Srinivasan_NeRV_Neural_Reflectance_and_Visibility_Fields_for_Relighting_and_View_CVPR_2021_paper.html`.

[145] J. C. Butcher and H. Messel. “Electron Number Distribution in Electron-Photon Showers”. In: *Phys. Rev.* 112 (6 Dec. 1958), pp. 2096–2106. ISSN: 0031-899X. DOI: `10.1103/PhysRev.112.2096`. URL: `https://link.aps.org/doi/10.1103/PhysRev.112.2096`.

[146] E. Woodcock, T. Murphy, P. Hemmings, and S. Longworth. “Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry”. In: *Proceedings of the Conference on Applications of Computing Methods to Reactor Problems*. 2. Argonne National Laboratory, 1965, p. 557.

[147] Markus Kettunen, Eugene D’Eon, Jacopo Pantaleoni, and Jan Novák. “An Unbiased Ray-Marching Transmittance Estimator”. In: *ACM Trans. Graph.* 40.4 (July 2021), pp. 1–20. ISSN: 0730-0301. DOI: `10.1145/3450626.3459937`. URL: `https://doi.org/10.1145/3450626.3459937`.

[148] Jan Novák, Andrew Selle, and Wojciech Jarosz. “Residual Ratio Tracking for Estimating Attenuation in Participating Media”. In: *ACM Trans. Graph.* 33.6 (Nov. 2014), pp. 1–11. ISSN: 0730-0301. DOI: `10.1145/2661229.2661292`. URL: `https://doi.org/10.1145/2661229.2661292`.

[149] Jean-Marc Tregan, Stéphane Blanco, Jérémi Dauchet, Mouna El Hafi, Richard Fournier, L. Ibarrart, P. Lapeyre, and Najda Villefranque. "Convergence issues in derivatives of Monte Carlo null-collision integral formulations: a solution". In: *Journal of Computational Physics* 413 (July 2020), p. 109463. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2020.109463.

[150] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (Dec. 2014). URL: http://arxiv.org/abs/1412.6980.

[151] H. Rief. "Generalized Monte Carlo perturbation algorithms for correlated sampling and a second-order Taylor series approach". In: *Annals of Nuclear Energy* 11.9 (Jan. 1984), pp. 455–476. ISSN: 0306-4549. DOI: https://doi.org/10.1016/0306-4549(84)90064-1. URL: https://www.sciencedirect.com/science/article/pii/0306454984900641.

[152] Iliyan Georgiev, Zackary Misso, Toshiya Hachisuka, Derek Nowrouzezahrai, Jaroslav Křivánek, and Wojciech Jarosz. "Integral Formulations of Volumetric Transmittance". In: *ACM Trans. Graph.* 38.6 (Nov. 2019), pp. 1–17. ISSN: 0730-0301. DOI: 10.1145/3355089.3356559. URL: https://doi.org/10.1145/3355089.3356559.

[153] Min-Te Chao. "A general purpose unequal probability sampling plan". In: *Biometrika* 69.3 (Dec. 1982), pp. 653–656. ISSN: 0006-3444. DOI: 10.1093/biomet/69.3.653. eprint: https://academic.oup.com/biomet/article-pdf/69/3/653/591311/69-3-653.pdf. URL: https://doi.org/10.1093/biomet/69.3.653.

[154] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz. "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 39.4 (July 2020), p. 148. ISSN: 0730-0301. DOI: 10/gg8xc7.

[155] Mark Lee, Brian Green, Feng Xie, and Eric Tabellion. "Vectorized production path tracing". In: *Proceedings of High Performance Graphics*. ACM. ACM, July 2017, 10:1–10:11. DOI: 10.1145/3105762.3105768.

[156] Alexander Keller, Carsten Wächter, Matthias Raab, Daniel Seibert, Dietger van Antwerpen, Johann Korndörfer, and Lutz Kettner. "The Iray Light Transport Simulation and Rendering System". In: *ACM SIGGRAPH 2017 Talks*. SIGGRAPH '17. New York, NY, USA: ACM, July 2017, 34:1–34:2. ISBN: 978-1-4503-5008-2. DOI: 10.1145/3084363.3085050. URL: http://arxiv.org/pdf/1705.01263.

[157] Brent Burley, David Adler, Matt Jen-Yuan Chiang, Hank Driskill, Ralf Habel, Patrick Kelly, Peter Kutz, Yining Karl Li, and Daniel Teece. "The Design and Evolution of Disney's Hyperion Renderer". In: *ACM Transactions on Graphics* 37.3 (July 2018), pp. 1–22. ISSN: 0730-0301. DOI: `10.1145/3182159`.

[158] Luca Fascione, Johannes Hanika, Marcos Fajardo, Per Christensen, Brent Burley, and Brian Green. "Path Tracing in Production - Part 1: Production Renderers". In: *ACM SIGGRAPH 2017 Courses*. SIGGRAPH '17. Los Angeles, California: ACM, 2017, 13:1–13:39. ISBN: 978-1-4503-5014-3. DOI: `10.1145/3084873.3084904`.

[159] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[160] Scott Meyers. *Effective C++: 55 specific ways to improve your programs and designs*. Pearson Education, 2005.

[161] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[162] Robin J. Hogan. "Fast Reverse-Mode Automatic Differentiation Using Expression Templates in C++". In: *ACM Transactions on Mathematical Software* 40.4 (July 2014), pp. 1–16. ISSN: 0098-3500. DOI: `10.1145/2560359`.

[163] Todd Veldhuizen. "Expression Templates". In: *C++ Report* 7 (1995).

[164] Bob Carpenter, Matthew D. Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. "The Stan Math Library: Reverse-Mode Automatic Differentiation in C++". In: *CoRR* abs/1509.07164 (2015). arXiv: `1509.07164`. URL: `http://arxiv.org/abs/1509.07164`.

[165] Eric Heitz and Eugene d'Eon. "Importance Sampling Microfacet-Based BSDFs using the Distribution of Visible Normals". In: *Computer Graphics Forum*. Vol. 33. 4. Wiley, July 2014, pp. 103–112. URL: `https://hal.inria.fr/hal-00996995/file/article.pdf`.

[166] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. "Embree: A Kernel Framework for Efficient CPU Ray Tracing". In: *ACM Transactions on Graphics* 33.4 (July 2014), 143:1–143:8. DOI: 10.1145/2601097.2601199.

[167] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. "OptiX: A General Purpose Ray Tracing Engine". In: *ACM Trans. Graph.* 29.4 (July 2010), pp. 1–13. ISSN: 0730-0301. DOI: 10.1145/1778765.1778803. URL: https://doi.org/10.1145/1778765.1778803.

[168] Samuli Laine, Tero Karras, and Timo Aila. "Megakernels Considered Harmful: Wavefront Path Tracing on GPUs". In: *Proceedings of the 5th High-Performance Graphics Conference.* Ed. by Kayvon Fatahalian, Christian Theobalt, and Jaakko Lehtinen. HPG '13. Anaheim, California: Association for Computing Machinery, 2013, 137–143. ISBN: 9781450321358. DOI: 10.1145/2492045.2492060.

[169] Jacob Mackay and David Johnson. "Millimetre wave ray tracing simulator with phase and beam effects using the Wigner distribution function". In: *Passive and Active Millimeter-Wave Imaging XXIV.* Ed. by David A. Wikner and Duncan A. Robertson. Vol. 11745. International Society for Optics and Photonics. SPIE, Apr. 2021, p. 1174508. DOI: 10.1117/12.2585861. URL: https://doi.org/10.1117/12.2585861.

[170] Christian Bischof, Alan Carle, George Corliss, Andreas Griewank, and Paul Hovland. "ADIFOR–generating derivative codes from Fortran programs". In: *Scientific Programming* 1.1 (1992), pp. 11–29. ISSN: 1058-9244. URL: http://downloads.hindawi.com/journals/sp/1992/717832.pdf.

[171] Laurent Hascoet and Valérie Pascual. "The Tapenade automatic differentiation tool: Principles, model, and specification". In: *ACM Transactions on Mathematical Software (TOMS)* 39.3 (2013), p. 20. DOI: 10.1145/2450153.2450158.

[172] Marios Papas, Wojciech Jarosz, Wenzel Jakob, Szymon Rusinkiewicz, Wojciech Matusik, and Tim Weyrich. "Goal-Based Caustics". In: *Computer Graphics Forum (Proceedings of Eurographics)* 30.2 (June 2011), pp. 503–511. ISSN: 0167-7055.

[173] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. "Poisson-Based Continuous Surface Generation for Goal-Based Caustics". In: *ACM Transactions on Graphics* 33.3 (June 2014), pp. 1–7. ISSN: 0730-0301. DOI: 10.1145/2580946.

[174] Yuliy Schwartzburg, Romain Testuz, Andrea Tagliasacchi, and Mark Pauly. “High-contrast Computational Caustic Design”. In: *ACM Transactions on Graphics* 33.4 (July 2014). Proc. SIGGRAPH 2014, pp. 1–11. ISSN: 0730-0301. DOI: 10.1145/2601097.2601200. URL: http://lgg.epfl.ch/publications/2014/Caustics/paper.pdf.

[175] Anurag Sharma, D Vizia Kumar, and Ajoy K Ghatak. “Tracing rays through graded-index media: a new method”. In: *Applied Optics* 21.6 (Mar. 1982), p. 984. ISSN: 0003-6935.

[176] Du T. Nguyen, Cameron Meyers, Timothy D. Yee, Nikola A. Dudukovic, Joel F. Destino, Cheng Zhu, Eric B. Duoss, Theodore F. Baumann, Tayyab Suratwala, James E. Smay, and Rebecca Dylla-Spears. “3D-Printed Transparent Glass”. In: *Advanced Materials* 29.26 (July 2017), p. 1701181. ISSN: 0935-9648.

[177] Arjun Teh, Matthew O’Toole, and Ioannis Gkioulekas. “Adjoint Nonlinear Ray Tracing”. In: *ACM Trans. Graph.* 41.4 (July 2022), pp. 1–13. ISSN: 0730-0301. DOI: 10.1145/3528223.3530077. URL: https://doi.org/10.1145/3528223.3530077.

[178] Cheng Sun, Min Sun, and Hwann-Tzong Chen. “Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction”. In: *arXiv:2111.11215* abs/2111.11215 (Nov. 2021). URL: https://arxiv.org/abs/2111.11215.

[179] A. Laurentini. “The Visual Hull Concept for Silhouette-Based Image Understanding”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 16.2 (Feb. 1994), 150–162. ISSN: 0162-8828. DOI: 10.1109/34.273735. URL: https://doi.org/10.1109/34.273735.

[180] Hailin Jin, Stefano Soatto, and Anthony J. Yezzi. “Multi-view stereo reconstruction of dense shape and complex appearance”. In: *International Journal of Computer Vision* 63.3 (July 2005), pp. 175–189. ISSN: 0920-5691. DOI: 10.1007/s11263-005-6876-7.

[181] Johannes Lutz Schönberger and Jan-Michael Frahm. “Structure-from-Motion Revisited”. In: *Proc. of CVPR*. IEEE. IEEE, June 2016, pp. 4104–4113. DOI: 10.1109/cvpr.2016.445.

[182] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. “Volume rendering of neural implicit surfaces”. In: *Advances in Neural Information Processing Systems* 34 (June 2021). Ed. by Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, pp. 4805–4815. URL: https://

proceedings.neurips.cc/paper/2021/hash/25e2a30f44898b9f3e978b1786dcd85c-Abstract.html.

[183] Delio Vicini, Wenzel Jakob, and Anton Kaplanyan. “A non-exponential transmittance model for volumetric scene representations”. In: *ACM Transactions on Graphics (TOG)* 40.4 (Aug. 2021), pp. 1–16. ISSN: 0730-0301. DOI: 10.1145/3450626.3459815. URL: https://dl.acm.org/doi/pdf/10.1145/3450626.3459815.

[184] Linjie Lyu, Ayush Tewari, Thomas Leimkuehler, Marc Habermann, and Christian Theobalt. “Neural Radiance Transfer Fields for Relightable Novel-view Synthesis with Global Illumination”. In: *arXiv preprint arXiv:2207.13607* abs/2207.13607 (2022). DOI: 10.48550/arxiv.2207.13607. URL: https://doi.org/10.48550/arXiv.2207.13607.

[185] Ben Mildenhall, Peter Hedman, Ricardo Martin-Brualla, Pratul Srinivasan, and Jonathan T. Barron. “NeRF in the Dark: High Dynamic Range View Synthesis from Noisy Raw Images”. In: *arXiv:2111.13679* abs/2111.13679 (Nov. 2021). URL: https://arxiv.org/abs/2111.13679.

[186] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. “KinectFusion: Real-time dense surface mapping and tracking”. In: *IEEE ISMAR*. IEEE, Oct. 2011, pp. 127–136. DOI: 10.1109/ismar.2011.6092378. URL: http://www.doc.ic.ac.uk/%7Eajd/Publications/newcombe_etal_ismar2011.pdf.

[187] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. “KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera”. In: *Proc. of ACM UIST*. Ed. by Jeffrey S. Pierce, Maneesh Agrawala, and Scott R. Klemmer. ACM, 2011, pp. 559–568. DOI: 10.1145/2047196.2047270.

[188] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. “Real-time 3D reconstruction at scale using voxel hashing”. In: *ACM Trans. Graph.* 32.6 (Nov. 2013), pp. 1–11. ISSN: 0730-0301. DOI: 10.1145/2508363.2508374. URL: http://graphics.stanford.edu/%7Eniessner/papers/2013/4hashing/niessner2013hashing.pdf.

[189] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration". In: *ACM Trans. Graph.* 36.4 (2017), p. 1. DOI: 10.1145/3054739.

[190] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. "The Replica Dataset: A Digital Replica of Indoor Spaces". In: *arXiv preprint arXiv:1906.05797* abs/1906.05797 (June 2019). URL: http://arxiv.org/abs/1906.05797.

[191] Paul Debevec. "Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography". In: *SIGGRAPH.* 1998, pp. 189–198. DOI: 10.1145/1401132.1401175.

[192] Robert Maier, Kihwan Kim, Daniel Cremers, Jan Kautz, and Matthias Nießner. "Intrinsic3d: High-quality 3D reconstruction by joint appearance and geometry optimization with spatially-varying lighting". In: *Proc. of CVPR.* IEEE. IEEE, Oct. 2017, pp. 3114–3122. DOI: 10.1109/iccv.2017.338. URL: http://arxiv.org/pdf/1708.01670.

[193] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. "Inverse Rendering for Complex Indoor Scenes: Shape, Spatially-Varying Lighting and SVBRDF from a Single Image". In: *Proc. of CVPR.* IEEE. IEEE, June 2020, pp. 2472–2481. DOI: 10.1109/cvpr42600.2020.00255. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/Li_Inverse_Rendering_for_Complex_Indoor_Scenes_Shape_Spatially-Varying_Lighting_and_CVPR_2020_paper.html.

[194] Marc-André Gardner, Kalyan Sunkavalli, Ersin Yumer, Xiaohui Shen, Emiliano Gambaretto, Christian Gagné, and Jean-François Lalonde. "Learning to predict indoor illumination from a single image". In: *ACM Trans. Graph.* 36.6 (Nov. 2017), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3130800.3130891. URL: http://arxiv.org/pdf/1704.00090.

[195] Mathieu Garon, Kalyan Sunkavalli, Sunil Hadap, Nathan Carr, and Jean-François Lalonde. "Fast Spatially-Varying Indoor Lighting Estimation". In: *Proc. of CVPR.*

IEEE. IEEE, June 2019, pp. 6908–6917. DOI: 10.1109/cvpr.2019.00707. URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Garon_Fast_Spatially-Varying_Indoor_Lighting_Estimation_CVPR_2019_paper.html.

[196] Marc-André Gardner, Yannick Hold-Geoffroy, Kalyan Sunkavalli, Christian Gagné, and Jean-François Lalonde. "Deep Parametric Indoor Lighting Estimation". In: *Proc. of ICCV*. IEEE. IEEE, Oct. 2019, pp. 7175–7183. DOI: 10.1109/iccv.2019.00727. URL: http://arxiv.org/pdf/1910.08812.

[197] Lixiong Chen, Yinqiang Zheng, Boxin Shi, Art Subpa-asa, and Imari Sato. "A Microfacet-based Model for Photometric Stereo with General Isotropic Reflectance". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 43 (Jan. 2019), pp. 48–61. ISSN: 0162-8828. DOI: 10.1109/tpami.2019.2927909.

[198] Kaizhang Kang, Zimin Chen, Jiaping Wang, Kun Zhou, and Hongzhi Wu. "Efficient reflectance capture using an autoencoder." In: *ACM Trans. Graph.* 37.4 (Aug. 2018), 127:1–127:10. ISSN: 0730-0301. DOI: 10.1145/3197517.3201279.

[199] Stephen Lombardi, Jason Saragih, Tomas Simon, and Yaser Sheikh. "Deep appearance models for face rendering". In: *ACM Trans. Graph.* 37.4 (Aug. 2018), p. 68. ISSN: 0730-0301. DOI: 10.1145/3197517.3201401. URL: https://dl.acm.org/doi/pdf/10.1145/3197517.3201401.

[200] Zexiang Xu, Kalyan Sunkavalli, Sunil Hadap, and Ravi Ramamoorthi. "Deep image-based relighting from optimal sparse samples". In: *ACM Trans. Graph.* 37.4 (Aug. 2018), p. 126. ISSN: 0730-0301. DOI: 10.1145/3197517.3201313. URL: https://dl.acm.org/doi/pdf/10.1145/3197517.3201313.

[201] Maxim Maximov, Laura Leal-Taixe, Mario Fritz, and Tobias Ritschel. "Deep Appearance Maps". In: *Proc. ICCV*. IEEE, Oct. 2019, pp. 8728–8737. DOI: 10.1109/iccv.2019.00882.

[202] Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. "Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images". In: *ACM Transactions on Graphics (TOG)* 38.4 (Aug. 2019), pp. 1–15. ISSN: 0730-0301. DOI: 10.1145/3306346.3323042. URL: https://dl.acm.org/doi/pdf/10.1145/3306346.3323042.

[203] Yue Dong. "Deep appearance modeling: A survey". In: *Visual Informatics* 3 (June 2019), pp. 59–68. ISSN: 2468-502X. DOI: 10.1016/j.visinf.2019.07.003. URL: https://doi.org/10.1016/j.visinf.2019.07.003.

[204] Carolin Schmitt, Simon Donné, Gernot Riegler, Vladlen Koltun, and Andreas Geiger. “On Joint Estimation of Pose, Geometry and svBRDF From a Handheld Scanner”. In: *Proc. of CVPR*. IEEE. IEEE, June 2020, pp. 3493–3503. DOI: 10.1109/cvpr42600.2020.00355. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/Schmitt_On_Joint_Estimation_of_Pose_Geometry_and_svBRDF_From_a_CVPR_2020_paper.html.

[205] Shen Sang and Manmohan Chandraker. “Single-Shot Neural Relighting and SVBRDF Estimation”. In: *ECCV*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Vol. 12364. Springer. Springer International Publishing, 2020, pp. 85–101. ISBN: 9783030585280. DOI: 10.1007/978-3-030-58529-7_6.

[206] Mark Boss, Varun Jampani, Kihwan Kim, Hendrik Lensch, and Jan Kautz. “Two-shot spatially-varying brdf and shape estimation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2020, pp. 3982–3991. DOI: 10.1109/cvpr42600.2020.00404. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/Boss_Two-Shot_Spatially-Varying_BRDF_and_Shape_Estimation_CVPR_2020_paper.html.

[207] Zhengqin Li, Zexiang Xu, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. “Learning to reconstruct shape and spatially-varying reflectance from a single image”. In: *ACM Trans. Graph.* 37.6 (Dec. 2018), pp. 1–11. ISSN: 0730-0301. DOI: 10.1145/3272127.3275055.

[208] Jonathan T Barron and Jitendra Malik. “Shape, illumination, and reflectance from shading”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 37.8 (May 2014), pp. 1670–1687. ISSN: 0162-8828. DOI: 10.1109/tpami.2014.2377712. URL: https://arxiv.org/abs/2010.03592.

[209] Kevin Karsch, Varsha Hedau, David Forsyth, and Derek Hoiem. “Rendering synthetic objects into legacy photographs”. In: *ACM Trans. Graph.* 30.6 (Dec. 2011), pp. 1–12. DOI: 10.1145/2024156.2024191. URL: http://arxiv.org/abs/1912.11565.

[210] Kevin Karsch, Kalyan Sunkavalli, Sunil Hadap, Nathan Carr, Hailin Jin, Rafael Fonte, Michael Sittig, and David Forsyth. “Automatic scene inference for 3d object compositing”. In: *ACM Trans. Graph.* 33.3 (May 2014), pp. 1–15. ISSN: 0730-0301. DOI: 10.1145/2602146. URL: http://arxiv.org/pdf/1912.12297.

[211] Harry Barrow, J Tenenbaum, A Hanson, and E Riseman. “Recovering intrinsic scene characteristics”. In: *Comput. Vis. Syst* 2 (1978), pp. 3–26.

[212] Abhimitra Meka, Maxim Maximov, Michael Zollhoefer, Avishek Chatterjee, Hans-Peter Seidel, Christian Richardt, and Christian Theobalt. "LIME: Live Intrinsic Material Estimation". In: *Proc. of CVPR.* IEEE. IEEE, June 2018, pp. 6315–6324. DOI: 10.1109/cvpr.2018.00661. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Meka_LIME_Live_Intrinsic_CVPR_2018_paper.html.

[213] Andrew Liu, Shiry Ginosar, Tinghui Zhou, Alexei A. Efros, and Noah Snavely. "Learning to Factorize and Relight a City". In: *ECCV.* Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Vol. 12349. Springer International Publishing, Aug. 2020, pp. 544–561. ISBN: 9783030585471. DOI: 10.1007/978-3-030-58548-8_32. URL: http://arxiv.org/pdf/2008.02796.

[214] Justus Thies, Michael Zollhöfer, and Matthias Nießner. "Deferred neural rendering: Image synthesis using neural textures". In: *ACM Transactions on Graphics (TOG)* 38.4 (Apr. 2019), pp. 1–12. DOI: 10.1145/3306346.3323035.

[215] Duan Gao, Guojun Chen, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. "Deferred neural lighting: free-viewpoint relighting from unstructured photographs". In: *ACM Transactions on Graphics (TOG)* 39.6 (2020), pp. 1–15. DOI: 10.1145/3414685.3417767.

[216] Darya Guarnera, Giuseppe Claudio Guarnera, Abhijeet Ghosh, Cornelia Denk, and Mashhuda Glencross. "BRDF representation and acquisition". In: *Computer Graphics Forum* 35.2 (May 2016), pp. 625–650. ISSN: 0167-7055. URL: http://spiral.imperial.ac.uk/bitstream/10044/1/31052/2/star1009_CRC_lowres.pdf.

[217] Brent Burley and Walt Disney Animation Studios. "Physically-based shading at disney". In: *ACM SIGGRAPH.* Vol. 2012. 2012, pp. 1–7.

[218] Dima Feldman and Yuval Shavitt. "An optimal median calculation algorithm for estimating Internet link delays from active measurements". In: *2007 Workshop on End-to-End Monitoring Techniques and Services.* Ed. by Kamil Saraç and Timur Friedman. IEEE. IEEE, May 2007, pp. 1–7. DOI: 10.1109/e2emon.2007.375318.

[219] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging". In: *SIAM journal on control and optimization* 30.4 (July 1992), pp. 838–855. ISSN: 0363-0129. DOI: 10.1137/0330046.

[220] David Ruppert. *Efficient estimations from a slowly convergent Robbins-Monro process.* Tech. rep. Cornell University Operations Research and Industrial Engineering, 1988.

[221] Diego Royo, Jorge García, Adolfo Muñoz, and Adrian Jarabo. "Non-line-of-sight transient rendering". In: *Computers & Graphics* 107 (Oct. 2022). Ed. by Munkhtsetseg Nandigjav and Olga Diamante, pp. 84–92. ISSN: 0097-8493. DOI: `https://doi.org/10.1016/j.cag.2022.07.003`. URL: `https://www.sciencedirect.com/science/article/pii/S0097849322001200`.

[222] Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. "Appearance-Driven Automatic 3D Model Simplification". In: *Eurographics Symposium on Rendering*. Ed. by Adrien Bousseau and Morgan McGuire. Eurographics Association, 2021, pp. 85–97. DOI: `10.2312/sr.20211293`.

[223] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. "Extracting Triangular 3D Models, Materials, and Lighting From Images". In: *arXiv:2111.12503* abs/2111.12503 (Nov. 2021). URL: `https://arxiv.org/abs/2111.12503`.

[224] Mathieu Galtier, Stéphane Blanco, Cyril Caliot, Christophe Coustet, Jérémi Dauchet, Mouna El Hafi, Vincent Eymet, Richard Fournier, Jacques Gautrais, Anaïs Khuong, et al. "Integral formulation of null-collision Monte Carlo algorithms". In: *Journal of Quantitative Spectroscopy and Radiative Transfer* 125 (Aug. 2013), pp. 57–68. ISSN: 0022-4073. DOI: `10.1016/j.jqsrt.2013.04.001`.

[225] Peter Kutz, Ralf Habel, Yining Karl Li, and Jan Novák. "Spectral and Decomposition Tracking for Rendering Heterogeneous Volumes". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)* 36.4 (July 2017), 111:1–111:16. ISSN: 0730-0301. DOI: `10.1145/3072959.3073665`.

[226] Bailey Miller, Iliyan Georgiev, and Wojciech Jarosz. "A Null-Scattering Path Integral Formulation of Light Transport". In: *ACM Trans. Graph.* 38.4 (July 2019), pp. 1–13. ISSN: 0730-0301. DOI: `10.1145/3306346.3323025`. URL: `https://doi.org/10.1145/3306346.3323025`.